ARMS: A DECENTRALISED NAMING MODEL FOR OBJECT-BASED DISTRIBUTED COMPUTING SYSTEMS

By

Jiabin Li, B.Eng. (Hons) (Curtin)

THIS THESIS IS PRESENTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY OF

MURDOCH UNIVERSITY

2010

DECLARATION

I declare that this thesis is my own account of my research and contains as its main content work that has not previously been submitted for a degree at any tertiary education institution.

_____

Jiabin Li

ACKNOWLEDGEMENTS

I would like to begin by expressing my immense gratitude and appreciate to my supervisors Associate Professor Dr. Lance Chun Che Fung and Dr. Douglas G. Myers for their advice, support, guidance and inspiration throughout the course of my study. Professor Fung deserves special thanks for all of his help, especially with the preparation of this thesis, his comments and interest in my work from when I was an Honours student until now. He is not only my mentor, but also a most valued friend whom I am forever grateful to have.

Many thanks to Associate Professor Dr. Kevin Kok Wai Wong and Dr. Kien-Ping Chung for their assistance during my time at Murdoch University. Thanks also to all the staff in the School of Information Technology, Murdoch University.

Finally, without the support of my parents and my wife, their understanding and patience, it would have been impossible for me to complete this study.

ABSTRACT

Entities communicate with one another in distributed computing systems via symbolic names. Implementing such communication requires a naming scheme that dynamically maps these symbolic names to physical nodes and processes. Traditionally, a centralised name server is deployed to perform such translations. However, a collaborative and dynamic environment requires a decentralised naming system due to reasons of efficiency and reliability.

ARMS (Adaptive, Randomised and Migration-enabled Scheme) is a novel decentralised naming scheme for distributed object-oriented computing systems. A notable feature of ARMS is that it provides direct naming supports for the patterns of object communication and object migration processes to achieve greater performance and scalability in executing object-oriented software within a distributed environment. These supports are driven by three key components: 1) *an adaptive locating protocol* that exploits the patterns of object communication and explores the best routing path in the face of the changing network conditions, 2) *a randomised overlay* that is a scalable and flexible substrate for routing name queries, and 3) *a hybrid relocation scheme* that provides a transparent and efficient means of referencing migrated objects.

The performance of ARMS has been examined using a number of real world Java-based benchmarking programs. Based on results in this study, ARMS has found to be superior to its structural counterpart – the *Chord model* because of the adaptive routing protocol and the resilient overlay. Furthermore, ARMS has shown to be superior in a number of other performance metrics.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATIONS

The following publications (in reverse chronicle order) were produced in the course of the research leading to this thesis. A taxonomy for classifying decentralised naming systems was first proposed in 2004 [9]. Subsequently, the focus of the work was on the design of a discrete-event simulation framework in 2005 [7,8] and a Java-based profiler in 2006 [6]. Results from the study of object locating mechanisms in an object-based distributed computing system were later reported in 2007 and 2008 [2-5]. Finally, the research of the transparent relocation scheme was published in 2010 [1].

[1] "*Migration Supports for Potential Mobile Objects in Ubiquitous Computing Systems*", Proceedings of the International Conference of Impact on Ubiquitous IT Co-Design to Industry, Perth, Australia, 21–23, January 2010.

[2] "*Application of Ant Colony Optimisation in Object Locating in A Distributed Computing System*", Proceedings of the Second IEEE International Conference on Digital Ecosystem and Technologies (IEEE-DEST 2008), 26-29 February, 2008, Phitsanulok, Thailand, ISBN: 1-4244-1490-3, pp 59-64.

[3] "*An Adaptive Randomised Structured Search Network for Locating Objects in a Distributed Computing System"*, Proceedings of the Eighth Postgraduate Electrical Engineering and Computing Symposium (PEECS 2007), Curtin University of Technology, Western Australia, November 2007, ISBN 1-74067-5673, pp. 17-22.

[4] "*Intelligent Object Locality Naming Model in An Object-based Distributed System for Engineering Applications*", Proceedings of the Sixth International Conference on Machine Learning and Cybernetics 2007 (ICMLC'07), Hong Kong, China, 19-22 August 2007, ISBN 1-4244-0972-1, pp 124-130.

[5] "*Efficient Locating and Relocating Scheme for Object-based Distributed Systems*", Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07), Las Vegas, Nevada, USA, 25-28 June, 2007, ISBN 1-60132-020-5, pp 69-75.

[6] "*Characteristics of Java Class File for Code Optimisation in a Distributed Computing Environment*", Proceedings of the 7th Postgraduate Electrical and Computing Symposium (PEECS 2006), ISBN 86905-977-7, Murdoch University, Perth, WA, pp. 13-17.

[7] "*Evaluation of a Small Scale Cluster Computing System for Parallel Intelligent Techniques Applications*", Proceedings of the Fourth International Conference on Machine Learning and Cybernetics (ICMLC 2005), ISBN 0-7803-9092-X, 19 - 21 August 2005, Guangzhou, China, pp 388-393.

[8] "*A Simulation Design for Evaluating Naming Models in Object-based Distributed Systems*", Proceedings of the 6th Postgraduate Electrical and Computing Symposium (PEECS '05), ISBN 0-7298-06090-X, Edith Cowan University, Perth, WA, pp. 166-171.

[9] "*Naming Models in Object-based Distributed Systems*", Proceedings of the *Fifth* Postgraduate Electrical Engineering and Computing Symposium (PEECS 2004), September 2004, Perth, Western Australia, pp 124-129.

1. Introduction

1.1. Naming Issues in Object-Based Distributed Computing Systems

The rapid increase in functionality and complexity of modern software systems has led to an alternative to the traditional procedural programming model – the Object-Oriented Programming (OOP) model [1]. Since its notable success in implementing the Graphical User interface (GUI) for modern computers, the OOP model has been supported by many programming languages [2-6] and has also been employed in numerous software applications [7-11]. However, the execution efficiency of OOP software remains a major concern due to the incompatibilities of the OOP model and the existing hardware structures. Different designs of hardware structures [12-23] have been suggested to specifically target at the OOP model. While those hardware structures have shown significant improvement on the execution efficiency, they largely focused on a sequential implementation. Only a few [12-14, 18] addressed the inherently distributed nature of the OOP model in their designs.

The OOP model relies on the simple concept of objects communicating with one another [3, 24]. An object is a self-contained entity that unifies its *state* and *methods* (or *code*) [1]. Thus, synchronisation is intrinsic and the concept of memory is distributed in the model. This in turn suggests that an ideal implementation should be based on a highly distributed system with efficient communication structures. Such a form of distributed systems is termed *object-based distributed computing system* (OBDCS) in this study.

In OBDCS, message passing is a means of communication between objects. A message is transported to the recipient object based on a *symbolic name* that is systematically

assigned to that object. On implementation, the symbolic name can be either dependent or independent with respect to the location of the associated object. Use of location independent names tends to be more favoured in practical distributed systems [25-27] as the communication layer within those systems is entirely abstract at the programming level. Such abstraction simplifies distributed programming and also permits different implementations without requiring to change the syntax of message passing, and so the application software [28-31]. Moreover, this abstraction is also attractive for object migration, and it is important for load distribution and reliability reasons. However, distributed systems with location independent names may incur a loss of computational efficiency due to the overhead of dynamically translating object names into their physical addresses. This forms the motivation of this thesis to investigate the efficiency issues in the dynamic name translation process, and how to resolve these issues within the context of OBDCS.

1.2. ARMS: an Efficient Decentralised Naming Model

The dynamic name translation is generally performed based on a set of *name-address mappings* managed by a naming system. A naming system can have a centralised [32, 33] or decentralised organisation [23, 34, 35]. A centralised organisation is however inadequate to support collaborative and highly dynamic environments [36, 37]. Furthermore, a central organisation suffers from performance bottlenecks and possibly a single point of failure when the size of the network becomes too large. A decentralised organisation is able to overcome these issues. However, using decentralised models [23, 34, 38, 39] can cause communication hotspots in the physical network [40] and undesirable overheads due to the mismatch between the system's decentralised organisation and the underlying network structure [41-43]. Other decentralised models such as Kazaa and Gnutella [36, 37] can potentially raise the efficiency and reliability issues. Finally, the designs of existing decentralised models have mainly targeted

services that provide distributed storage over a wide area network, particularly at the scale of the Internet. Most issues that have been tackled by those models are not entirely relevant to object communication in OBDCS. Consequently, it is desired to design a decentralised naming model that adapts to runtime object communication and network load status.

ARMS (*Adaptive, Randomised Migration-enabled Scheme*) is a novel decentralised naming model aimed at providing direct naming supports for distributed object oriented computing systems. In contrast to the existing naming models, ARMS focuses on an adaptive method based on the *ant colony optimisation* technique (ACO), to exploit message passing between objects. The adaptive method allows efficiently translating object names into addresses in the face of the changing topology of object relationship at runtime. The dynamic object relationship led to the varied patterns of message passing as shown in the results from this study. Those patterns were analysed in order to finetune the performance of the adaptive method and they are reported in Chapter 4. For the purpose of this study, it is reasonable to consider just one OO programming language as the focus is based on the general characteristics of the OOP model rather than a specific OOP implementation. Due to its importance and popularity, Java was chosen [3].

Furthermore, the decentralised organisation of ARMS is built upon a *randomised overlay* that places a randomised graph over a structured topology. The randomised overlay essentially helps to reduce hotspots in the physical network and to minimise communication overheads caused by the mismatch between the overlay topology and the underlying network structure. To further improve the effectiveness of the

randomised overlay, *heuristic rules* have also been examined for optimising the structure of the overlay at runtime and they are discussed in Chapter 6.

In modern distributed systems, *migration* – the act of transferring computational or data objects between machines, is the key to resource management. Migration has the potential to improve the performance, reliability and scalability of a distributed computing system as it enables dynamic load distribution, fault resilience and access locality [44, 45]. When an object migrates, it is essential that the object is still accessible by the same name and mechanism, as if the migration has never occurred. In this thesis, ARMS proposes a *transparent relocation* scheme for this reason. The transparent scheme seamlessly integrates three widely used relocation techniques – *the home-based approach*, *the forward pointers* and *the backtracking approach* to assist the process. Hence, the essence of the ARMS model is an ant-inspired routing scheme, a randomised overlay, and a transparent and lightweight relocation scheme addressing the naming issues in object-based distributed computing systems.

1.3. Thesis Contributions

Contributions of this work to the discipline can be summarised in the following five major areas.

- *A novel naming model specifically directed at object-based distributed computing systems has been proposed.* The model features adaptability, efficiency and scalability through an integration of an adaptive locating protocol and a randomised P2P network. This study revealed that the adaptive protocol improved distributed name resolution under different performance measures.

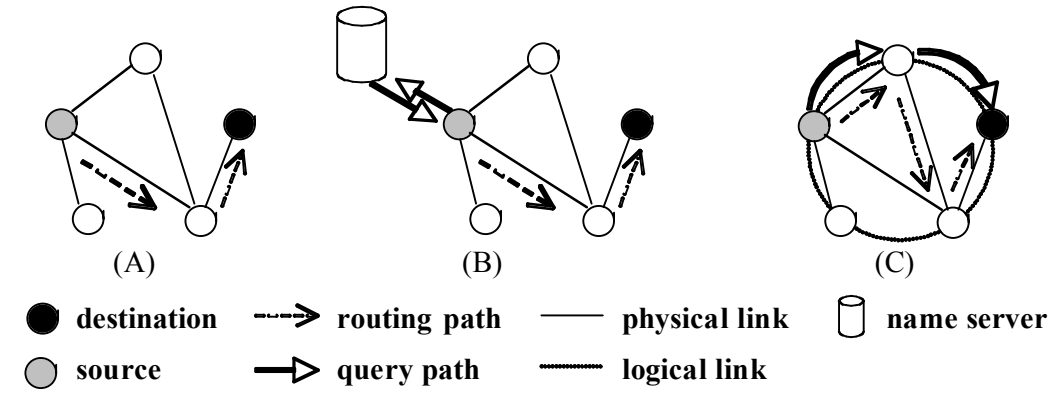

Figure 1: An illustration of the three forms of message routing: (A) routing based on location dependent names; (B) routing with location independent names via a name server; (C) routing directly based on location independent names.

- *A hybrid relocation scheme that seamlessly integrates with the adaptive locating protocol has been proposed.* The relocation scheme was designed to support transparent object migrations. The scheme combined the advantages of three common relocation models: *home-based, forward pointers* and *backtracking*.

- *Routing of objects based on location-independent names has been examined based on a combination of the naming and the routing processes as illustrated in Figure 1*. This proposal not only improves the communication throughput but also simplifies the implementation of a communication system.

- *Examinations of various kinds of constructive heuristic information have been carried out.* The heuristics were designed to tune the performance of the decentralised name translation and they can be used as guidelines for the design of the object oriented distributed computing systems.

- *A study of the patterns of object communications was conducted based on a set of Java benchmark programs.* Results from the study and in particular, knowledge about the patterns of object execution will be beneficial for continuous research of distributed object-oriented computing systems.

1.4. Thesis Structure

The thesis starts by defining the naming issues in object-based distributed computing systems. Study of these issues then led to the development of the ARMS naming system in this thesis.

Chapter 2 reviews existing naming systems. It first introduces the naming models, and followed by a summary of hierarchical centralised naming systems used in distributed systems. The chapter then reviews four classes of decentralised naming systems and their concepts that formed the basis of the development of ARMS.

Chapter 3 describes the key contribution of this thesis as related to ARMS. It describes the architectural design of ARMS, which includes a randomised overlay network, an adaptive locating algorithm, and a hybrid relocation scheme. Furthermore, the correctness of the locating protocol and the relocation protocol was verified using the *finite state machine (FSM)* theory.

Chapter 4 focuses on the finding from recorded patterns of object communication. This formed the basis for the development of constructive heuristic rules in order to improve the efficiency of the dynamic name translation for ARMS. The chapter begins with a summary of methodology used in this study, and then it describes the patterns of object communication, which were obtained by profiling a set of Java-based benchmark programs.

Chapter 5 studies the characteristics of the proposed adaptive locating protocol. The adaptive protocol was tested under various parameters used by the ACO algorithm. Furthermore, the scaling properties of ARMS are also discussed in this chapter.

Chapter 6 evaluates the performance of two sets of constructive heuristic information. The first set was developed based on the patterns of object communication. The second set was associated with the system properties.

Chapter 7 focuses on the performance aspects of the hybrid relocation scheme when it is used to resolve the new addresses of migrated objects. Two relocation schemes – home-based and forward pointers – have been used as benchmarks in the study.

Finally, Chapter 8 draws the conclusions from this study and discusses future directions for research in object-based distributed computing systems.

2. A Study of Dynamic Naming Systems

2.1. Overview

To name a software entity or a hardware resource by a symbolic name, is to describe them by an ***abstraction***. It means that the actual entity can be changed without affecting the overall software systems design. For example, a new printer should be able to be installed easily in a system without any need to change the application software. The value of such abstraction was recognised many years ago [28-31, 46]. If an entity is frequently relocated within a computer structure, as it commonly would be for load shedding reasons in a distributed computing system, then a ***dynamic naming system*** is required. The entity's name can remain the same, although the internal mechanisms need to track the entity's actual location. Such a naming system is vital as abstraction offers the advantage that software developers do not have to be concerned with entity locations, and similar issues dealing with load shedding and object migration.

This chapter gives a survey of the existing dynamic naming systems within the context of distributed systems. It is important to clarify that these distributed systems are not necessarily designed for object-oriented computing systems**.** However, it is useful to consider them in order to draw ideas to address the key issues such as, scalability, reliability and object migration related to OBDCS.

2.2. Name Conventions

Names are created based on a group of syntax rules governed by a ***Naming Model****.* A naming model is a specification that defines the name convention, the binding rules and the resolution process over a set of symbolic names [47]. A naming system is a particular instance of a naming model.

A name convention defines how names are generated from symbolic characters. There are three general rules for name constructions: *flat names*, *partitioned names* and *descriptive names* [31].

Flat names are those having no internal structure. They are generally not human readable and are mainly designed for machine uses. They are simple for implementations, as they do not impose any requirements on the structure of the naming system. Furthermore, the simple structure is also efficient for name resolution [30]. Due to these advantages, flat names have been widely adopted in wide-area networking systems [23, 34, 38, 39]. They will continue to play an important role in future Internet applications [48].

Partitioned names are generated based on well-defined syntax rules. Each partition effectively defines a context or a local name. A naming context is the domain in which the local name is bound [49]. A name in this category is commonly formed by composing contexts that are structured in a hierarchy. Naming systems based on partitioned names are termed the *contextual naming system* since their design focuses on context management. These systems are suitable for naming resources in a dynamically emerging environment such as the Internet. While the majority of naming systems being used today are in this category, naming systems derived from flat names or descriptive names are on the rise.

Descriptive names have a more complicated structure than the previously mentioned structures. They contain a list of attributes that describe the denoting entity. Each attribute comprises an attribute type and an attribute value. These attributes may or may

not be organised in order. Naming systems designed for descriptive names can apply the attributes in resolution via a pattern-matching scheme, instead of a simple mapping function that is used to resolve flat or partitioned names. An example of these naming systems is a trading system operated in a CORBA platform [50]. A partitioned name, nonetheless, can be considered as a special form of a descriptive name that includes a rigid set of orderly attributes.

2.3. Binding and Resolution

Binding and *resolution* are two complementary mechanisms. Binding establishes the mappings of names to entities. Resolution subsequently accesses the entities, as well as their associated attributes such as their IP addresses, by resolving names based on the mappings. Corner and Peterson [51] have observed that the name resolution is a strictly syntactical process. The semantic information resides in the binding of names to entities and such information is invariant to a resolution mechanism. A name resolution, however, is effectively a translation process that confines to the predefined binding rules. Bayerdorffer [47] used program compilation to illustrate the relationship between binding and resolution. Compilation of programs, where syntactic operations are translated into corresponding operations, relies on a language definition, which defines mappings of operations to syntactic operations. Research [29, 47, 51] has shown that properties of name resolutions have to exclusively deal with the requirements of an implementation for the naming model. As far as a distributed system is concerned, one of the key requirements is to consider the characteristics of communication among entities.

Saltzer [46, 52] is amongst the early researchers [28-31, 46] who conducted systematic studies for location-independent names. Saltzer presented a layered naming architecture for entities used in a data communication network. There are four types of entities:

services, nodes, attachment points and *routes*. The scheme is an extension of the Shoch's definition of *names, addresses* and *routes*. The notable feature of the binding scheme is its independent and changeable structure. Names of entities in different categories are managed independently and therefore bindings at one layer can be changed without affecting another layer. The concept of independent bindings is still pertinent to naming systems for mobile and distributed object systems [49]. Furthermore, Saltzer proposed a classification model based on the purposes of names – human versus computational use and local versus universal names. However, the classification model is rather simple, and more, it is incomplete.

Bayerdorffer [47] has observed that patterns of communication are essentially information flows from a source entity to a set of destination entities based on a set of names. Hence, the different designs of binding rules can lead to the distinct patterns of communication. Through the concept, Bayerdorffer developed a taxonomy model for naming systems based on a set of orthogonal properties of bindings:

- *Mutability of bindings* – determines the ability of systems to alter bindings at runtime;

- *Domain knowledge* – characterises the ability of name domains to learn new names during a computation;

- *Multiply-bound names* – a name can be simultaneously bound to a single entity or multiple entities;

- *Multiply-named entities* – an entity can be referred through aliased names;

- *Name sharing* – determines whether a name is accessible by multiple entities; and

- *Descriptive names* – a name can include attributes of the denoting entity.

Based on the taxonomy, a naming system that provides for all these attributes is represented as $\top$ while a naming system has none of the defined attributes is $\perp$. Bayerdorffer's work has influenced a number of distributed systems including DISCWorld [49], SINA [53], and a Desktop Grid system [54]. Further, the classification scheme of Falkner [49] extends the Bayerdorffer taxonomy to include existing ORB (Object Request Broker) models and mobile object systems.

A name resolution returns useful information for constructing a route to the entity that is bound to that name. Functions of name resolutions can be categorised into two classes – *non-descriptive* or *descriptive*.

Non-descriptive naming systems handle simple form of names like flat names and partitioned names. These systems normally require a lookup table that stores bindings of names to entities. Hence, they are also known as the lookup system [55]. Early researchers of lookup systems include Shoch [30], Postel [32], Hauzeur [31], Mockapetris [33], and Saltzer [52].

To help the study of Internet-wide computer networks, Shoch [30] made a distinction between three types of identifiers – *names, addresses* and *routes* – used in mapping. Such a distinction is important for supporting resource migration. A name needs not be

mapped to an address until a mapping function occurs. Hence, the address can be varied over time without affecting the name. This is the basic principle of a relocation scheme utilised in mobile object systems. However, Shoch's concept of addresses is too ambiguous and thus Hauzeur [31] has developed a more comprehensive model. Hauzeur first argues that addresses are essentially intermediate names and are used to ease the process of mapping names to routes. He proposed a multiple-layered naming model largely inspired by the Open Systems Interconnection (OSI) model [56].

In Hauzeur's model, names are identifiers at the top layer and they are translated into addresses. These addresses, however, are also identifiers at the lower layer and they are further translated into more concrete addresses. Therefore, a route can be considered as a list of addresses that are expanded by passing through sub-layers [31]. The new model subsequently leads to two types of mapping processes: *indirect mapping* – where a name is first mapped to an address and then the address is mapped into a route, and *direct mapping* – where a name is directly mapped to a route. While most distributed systems exercise an indirect mapping process, the concept of direct mapping opens a new routing paradigm – location-independent routing [34]. In particular, location-independent routing has been the prime focus in peer-to-peer (P2P) systems [23, 38, 39].

On the other hand, descriptive naming systems are closer to a database system. Descriptive naming systems provide functionalities that are more elaborate than non-descriptive systems because they are capable to return approximate answers to imperfect client requests [49, 55]. The resolution function of descriptive systems is done through matching a collection of attributes that is submitted by clients with a set of records that is defined in the database. Design of the pattern-matching approach

encounters two considerations: *which attributes should be matched* and *the ordering of attributes applied in matching*.

Bowman *et al.* [55] have studied naming systems as specialised inference mechanisms and subsequently proposed a preference hierarchy that is used to specify the resolution functions associated with a given naming system. A preference is a total order on a set of approximate functions, which are used to choose attributes. The preference hierarchy is useful for not only categorising existing naming systems but also for designing new naming systems. The model led to two new attribute-based naming systems: respectively, Univers [57] and $\text{Profile}_{\text{new}}$ [55]. Particularly, the $\text{Profile}_{\text{new}}$ system is an improved system as the result of correcting the flaws in the original Profile [58] under the guidance of the preference hierarchy.

Because the two classes of naming systems are substantially different in design and implementation, this study primarily focused on lookup systems designed for large-scale distributed object-oriented computing systems. While it is straightforward to implement lookup systems in small-sized systems, implementations remain complicated for big and dynamic networks due to the concerns of efficiency [34, 59], scalability [33] and load balancing [60].

2.4. Hierarchical Lookup Systems

The early distributed computing systems that were built for the DARPA network [61] relied on a local file, termed HOSTS.TXT, to map host names to their addresses. However, the HOSTS.TXT files failed at the rapid growth of the network size because they simply became too large to maintain [33]. Hence, distributed lookup systems have been examined to address this issue.

A distributed lookup system is a distributed implementation of a naming model for provision of name resolutions for a computer network. In the lookup system, a name space is partitioned and administrated collaboratively by the member nodes of the naming system. A name space represents a set of names that are generated from a naming convention. Each member node stores a subset of names in the name space. Such a node is termed the *authoritative* node and is responsible for binding and resolving names within the name subset [62]. With a distributed lookup system, the name resolution is essentially to locate the authoritative node of a given name.

The architectures of distributed lookup systems designed for Internet applications have been traditionally adapted from a tree structure. Every node from such a tree represents a context. Consider a context is a domain in which names are valid. Therefore, partitioned names, where contexts are explicitly specified in a name, are mostly used in those lookup systems. Despite various implementations of hierarchical lookup systems, the name resolution procedure is more or less similar. It begins with a root node and progresses downwards through branches. For example, an Internet domain name – *it.murdoch.edu.au* – is interpreted from right to left iteratively through these context nodes: a root node → the ***au*** context node → the ***edu*** context node → the ***murdoch*** context node. Finally, the local name ***it*** is resolved under the ***murdoch*** authoritative node.

The server/client model has been commonly used to implement distributed lookup systems. In this model, an authoritative node is realised by a contextual name server. The name server is responsible for answering name queries or requests submitted by clients [32]. The characteristics of these name servers include

- *Persistency* – name servers generally have fixed network addresses and hence this ensures the high availability of name services;

- *Centralised management* – an organisation can manage its name domain, such as adding new names to the domain, through the direct control of the local name server; and

- *Heterogeneity* – the separation of implementations of servers and clients enables the incorporation of a variety of system and network implementations.

A range of name servers has been designed for obtaining information on different entities: network hosts, distributed objects, and files. Name servers that administrate identifiers of network hosts are termed *domain servers*. Notable domain servers include Internet Name Server [32], Berkeley Internet Name Domain [62], and Domain Name System [63, 64, 65]. Domain servers are a compulsory component of the Internet today, as numerous applications such as World Wide Web (WWW), Email and Cloud computing [153] have to rely on name services provided by these systems. However, the rise of Internet-wide distributed computing like Grid computing and peer-to-peer file sharing has revealed the limitations of domain servers. Ineffective caching, centralisation, and inadequate support for services and data are among the major problems encountered in domain servers [48, 60, 66].

Clients of a domain name system largely read information from the system and they are normally not allowed to alter the name context. As a result, domain name systems have been fine tuned for read-only operations and caching. Distributed computing systems, however, require a naming system that facilitates context updating, entity binding and

unbinding, and relocating. Consequently, a number of *directory systems* have been proposed for various distributed computing frameworks such as CORBA [25], Java RMI [26], Globe [67], Microsoft .NET Remoting [27] and DISCWorld [49].

CORBA, Java RMI and .NET Remoting are distributed object systems that are largely influenced by the architecture of DCE (Distributed Computing Environment) [68]. Furthermore, object communication supported by those systems is based on a variant of the RPC (Remote Procedure Call) [69] mechanism.

The Common Object Request Broker Architecture (CORBA) [25, 50, 70] is a specification for an ORB architecture to provide language-neutral support for the server/client computing model. Object Request Broker (ORB) is an intermediate layer that facilitates remote procedure invocations. In addition, ORB acts as a simple naming agent that 1) obtains an IOP (Interoperable Object Reference) based on a symbolic name through some form of naming services, and 2) translates that IOP into a remote reference to the target object. An IOR is a unique name for a CORBA object and it consists of information about the object and specific protocol details.

A client can obtain an IOP through one of two OMG-specified naming services: COS (Common Object Service) naming service [70] and trading object service [50]. Similar to a domain name service, a COS naming service organises a name space hierarchically. The name space can contain context bindings to other COS naming servers. The joint space is also known as a *federated name space* [70]. Because there is an absence of a "universal" root in a federated name space, a client needs to know the location of an initial naming context for the resolving name. Alternatively, CORBA supports a trading service [50] to enable clients to use semantic descriptions in the name lookup. This

facilitates the offering and the discovery of services available from CORBA application servers. Moreover, a client may interact with the resolution process by setting appropriate attributes.

Java Remote Method Invocation (RMI) [26] specifies protocols for referencing Java remote objects. Communication between remote objects is handled by RMI stub objects and skeleton objects. A local registry, which manages bindings of names to objects, is used for the lookup of remote objects. However, Java RMI does not explicitly specify a hierarchy or grouping of registry systems and thus this limits support for distributed naming services. As a result, a Java-specified naming service [71, 72], termed Java Naming and Directory Interface (JNDI), has been designed for provision of distributed naming services and discovery services [71].

Similarly, Microsoft .NET Remoting [27] is another implementation of RPC that targets Windows-centric distributed systems. Remote procedure invocations are conducted through proxy objects. Names of remote objects are based on the URI (Uniformly Resource Identifier) scheme and hence names can be easily translated into references by proxy objects. These names are stored and managed by Active Directory [73], a technology created by Microsoft that supports extensive naming services including LDAP [74] and DNS. LDAP (Lightweight Directory Access Protocol) is an alternative protocol for accessing X.500 [75] directory services – attribute-based lookup through a widespread TCP/IP stack.

Globe [67], Globus [76, 77], Legion [78, 79, 80] and DISCWorld [49] are distributed object systems developed towards grid computing. These systems are designed for a geographically distributed, heterogeneous, and highly dynamic environment. Thus,

naming services implemented in a grid system must incorporate with a large-scale name space.

Globus is a grid infrastructure that encompasses Grid Security Infrastructure (GSI), Globus Resource Allocation Manager (GRAM), Metacomputing Directory Service (MDS), Global Access to Secondary Storage (GASS), data catalogue and replica management, and Advanced Resource Reservation and Allocation (ARRA). In particular, Metacomputing Directory Service (MDS) [81] is an extension to the LDAP scheme aimed at efficient and scalable accesses to diverse, dynamic, and distributed information about the structure and the state of resources. Names are organised within the MDS as a directory information tree (DIT) in order to simplify the process of the name lookup. The construction of the directory information tree is analogous to the URI scheme. However, these names are resolved using the LDAP mechanism for attribute matching and searching [81]. Grid projects based on the Globus include MPICH-G [82, 83] and Cactus [84].

Unlike other coarse-grained object systems like CORBA and Globus, Legion is an object-based grid system that supports concurrent processing with large degrees of parallelism. Objects of the Legion are the primitive units in computation and they can represent any hardware and software components. The uniform model of objects can provide users with a single, coherent, virtual architecture constructed from geographically distributed and heterogeneous machines. A reference of a Legion object is obtained via two stages of name resolutions [79]: 1) the symbolic name of this object is translated into an object ID (LOID) via context objects; 2) the object ID is subsequently resolved to a reference by binding agents. Binding agents can be organised into a hierarchical structure with a local root termed the Legion class. These

binding agents can cache bindings to achieve good lookup performance. Opposite to CORBA or RMI, the Legion provides a single and persistent namespace for ease of developing applications for a wide-area system [80]. However, LOIDs are not location transparent and thus object migration is not natively supported by the Legion.

Globe is an object-based grid system designed to provide a unifying paradigm for constructing wide-area distributed systems. The notable features of the Globe are the distributed shared object model and location transparent handlers. Communication is performed via distributed shared objects, where multiple processes can simultaneously access to the same object. The state of a distributed object is physically distributed across different address spaces and hence a mechanism is required to propagate any change of the state made by a process to other distributed objects. The concept of distributed shared objects is rather similar to PVM (Parallel Virtual Machines) [85].

In the Globe, a distributed object is constructed from a group of local objects resided in separate address spaces, where a local object is an implementation of an interface of the distributed object. A process needs to perform a binding process before it can communicate through a distributed object. The result of the binding is that a local object is placed in the address space of the requesting process, as analogised to the construction of a proxy object in the .NET Remoting. A binding involves two stages – object finding and interface installation. The Globe implements a two-level naming service: 1) a symbolic name is mapped to an object handler – a unique and location transparent ID, and 2) the object handler is resolved into a context address that describes a network address of an object and a communication protocol. The advantages of object handlers include support of multiple names for an object and convenience for object migration. A similar naming service has also been utilised in DISCWorld, which

is a geographical information system. However, DISCWorld implements a state-based ORB model for accessing to name services and communicating remote objects.

2.5. Decentralised Naming Systems

The hierarchical lookup systems are currently the de-facto naming system used in distributed computing systems based on the server/client architecture, such as Cloud computing [153]. Despite their popularity, the static central servers raise efficiency and reliability issues. The hierarchical structure can cause hotspots in name lookup as a recent study [86] showed over 90% of domain names were served by three or less DNS servers. The hotspots can seriously harm the efficiency of the name resolution and the balance of the workloads within the network [86]. In addition, failures at the top-level servers of the hierarchical system can potentially paralyse the whole system due to the highly dependent architecture.

The rapid expansion of the large-scale decentralised computing such as grid computing and file sharing demands more liberated and collaborative naming services. In these decentralised computing systems, computing resources and services are shared by directly exchanging between machines. Thus, a centralised control or structure is not compulsory for these systems. Moreover, the decentralised architecture can effectively prevent the single point of failure or bottlenecks. Hence, this architecture can offer better scalability.

There has been much interest in the emerging peer-to-peer (P2P) models that do not impose any central control or structure. The peer-based models are distributed in nature and they are well suited for constructing massively scalable, self-organising and robust systems. While the concept of P2P is not new, the concept has been emphasised more recently due to the demand of resource sharing applications over the Internet [87, 88].

In P2P systems, nodes or peers have symmetric in roles where they are both clients and servers. Each peer manages local contents and shares its contents with a subset of the other peers, which are known as logical *neighbours* of that peer. These logically connected peers form a *virtual topology* on top of a physical network. Such a virtual topology, also known as an *overlay network*, is the basis of search in peer-based systems. A routing table, sometimes called *forwarding table*, provides the translation of logical links to the physical addresses of peers. Moreover, P2P systems explicitly or implicitly provide a lookup mechanism, or locator function, for the name lookup.

Upon the name lookup, the requesting peer sends name queries to its immediate neighbours on the overlay network. If one of these neighbours is able to answer the name query, the address of the requested entity is returned to the requesting peer and the lookup process is terminated successfully. Otherwise, the name queries will be forwarded to the neighbours of the neighbours until the request is fulfilled or failed.

Although they have been typically designed for file sharing over the Internet [87, 88, 89], P2P models have been also actively researched in other areas such as grid computing [54] and sensor networks [53] because of its unique characteristics [90]

- *Resource sharing* – P2P systems facilitate resource sharing. These resources include physical resources such as CPU and storage or logical resources such as services;

- *Decentralisation* – Peers work collaboratively since there is no central coordination in a P2P system. The loads of the system are shared among peers.

This can reduce the risk of a single point of failure and improve the utilisation of system resources;

- *Scalability* – Another benefit of the decentralisation is improved scalability. P2P systems can easily scale to millions of nodes [36, 37, 88] due to the intrinsically distributed nature;

- *Autonomy and self-organisation* – Due to the lack of a central coordination, peers are capable of behaving autonomously, including joining of an existing network and responding to service requests. The autonomy of P2P models can consequently lead to a more flexible and reliable structure;

- *Dynamism* – P2P systems are inherently distributed and dynamic. The behaviour of the entire system is determined by a collection of states of the individual peers and how they interact with each other. This is well suited to a constantly changing environment such as Internet; and

- *Location-independent routing* – Some P2P systems seamlessly integrate a lookup protocol with a routing protocol. Thus a request message, which embeds the symbolic name of the requesting entity, is directly routed to the peer that possesses the requesting entity, without the need of knowing the physical address of that entity. The location-independent routing is desired for any large-scale network application.

Early studies [90, 91, 92] have classified P2P systems simply based on the means of constructing the overlay network. Hence, there are two basic classes of P2P overlay

networks – *Structured* and *Unstructured*. A structured overlay network [23, 34, 38, 39] implies that the overlay topology is tightly controlled and contents are placed at a specified location. Such structured P2P systems are commonly built by using the Distributed Hash Table (DHT) as a substrate. An entity is assigned to a unique identifier called *key*. A subset of keys is mapped by the overlay network to an exclusive peer. DHT-based systems can subsequently support retrieval of {*key*, *value*} through lookup operations, which involve routing queries to the peer corresponding to the key. The scalability and efficiency of the lookup operation can take an advantage of the restricted data placement so that any entity can be located using a small overlay hops – *O(logN)* on average, where *N* is the number of peers used in the system.

On the contrary, unstructured overlay networks [87, 93-97] are ad-hoc in nature; they do not require any prior knowledge of the topology and thus peers are connected based on a loose rule. These systems typically apply lookup function using time-to-live (TTL) controlled flooding mechanisms. Upon name resolution, a requesting node broadcasts query messages to all known neighbours. The neighbours either rely to the requesting node if the querying entity is locally available or forward the query messages to their neighbouring peers. The unrestrained topology organisation can cause redundant paths established between peers, making unstructured overlay networks resilient to node or link failures. Moreover, the flooding-based approach directly supports keyword-based searching. While this can also be provided by structured P2P systems, complex queries have to be done via an extra layer on top of the structured overlay. Consequently, unstructured P2P systems are still the predominant form of peer-based systems over the Internet today [36, 92, 98].

Although unstructured overlay networks can simplify implementations, it is obvious that the flooding-based mechanism does not scale well because the load on every node grows exponentially with the total number of queries and the network size [92]. Consequently, new models [99-103] have been proposed to achieve scalable entity retrievals through a *Constrained* overlay network. Similar to structured overlay networks, constrained P2P systems design an efficient and compact routing table on every peer. Therefore, constrained systems can effectively reduce the number of messages required in query forwarding. However, those systems do not imply a regular overlay topology and hence they do not necessarily guarantee that all contents are reachable by the locating mechanism.

On the other hand, several P2P systems [35, 41, 104, 105, 106] have been developed based upon a randomised graph topology over a structured overlay. This idea is inspired by observations of *the Small-World Phenomenon* [107] [101] in social networks. The routing aspect of the Small-World Phenomenon was originally described in [107] by Jon Kleinberg. The graphs, termed *randomised network*, not only have a small network diameter but also allow short routes to be discovered based on the local information. Furthermore, the flexibility of the randomised overlay structure can resolve the *Topology Mismatch problem* [41, 42, 43]. The problem arises from the difference of an overlay topology and a physical network structure. Subsequently, randomised systems can route queries more effectively via a route along peers that are physically closed together. Such a routing approach is also known as proximate routing [41, 43].

2.6. Classification of Peer-Based Naming Systems

2.6.1. Ad hoc routing on unstructured overlay (ARUO)

Existing P2P systems can be categorised based on two key elements: *overlay topology* and *routing scheme*. The overlay topology describes an architecture of virtual networks, which determines the organisation of peers and how they are logically connected. There are four classes of overlay topologies: *unstructured*, *structured*, *constrained*, and *randomised*. The routing scheme, alternatively, illustrates the object lookup protocol and the query routing protocol over the overlay network. There are also four classes of routing schemes: *ad hoc*, *static*, *informed*, and *proximate*.

Unstructured overlay models were developed for the early generations of P2P systems. In this category, a loose overlay structure is specified where peers are organised in a random manner. Consequently, there are minor overheads incurred in peers arriving at or departing from the network. This model is thereby well suited to support transient peers, where the connection period of those peers is generally short. Because not only peers have no a-priori knowledge of the network topology but also the topology is constantly evolving, ad hoc routing protocols like *flooding* are generally applied for discovering unstructured networks. Flooding is essentially a Breath-First-Search (BFS) algorithm, where a search explores *all* nodes of a graph, beginning at the source node.

Gnutella [37, 87] is a decentralised protocol for distributed search over an unstructured overlay network. To locate a resource, a Gnutella peer, or servant, floods queries to its neighbours within a certain radius controlled by a Time-To-Live (TTL) counter. In addition, the address of the querying Gnutella node is unknown to the replying Gnutella node. Hence, the replied message is sent along the reverse path travelled by the query message. The replying scheme is particularly designed for providing anonymity for the

querying node while it uses more aggregate network bandwidths. Obviously, the flooding-based approach is extremely costly and leads to poor scalability and long responding time. Furthermore, the flooding causes duplicated messages being received by the nodes in the neighbourhood. Those messages do not improve the probability of finding the requested entity; they can cause network congestions and hence block other useful communications. Consequently, the *hit rate*, which is inversely proportional to the total number of sent queries, is extremely low with a flooding-based approach.

Several ideas have been examined in Gnutella-like P2P systems [95] aiming at improving the efficiency and the scalability of the flooding-based search. Yang and Hector [95] evaluated the effectiveness of routing protocols based on two metrics; 1) *Cost of queries* where it is described in terms of bandwidths and processing power; and, 2) *Quality of results* where it is described in terms of the size of the total result set, satisfaction of the query, and time to satisfaction.

Subsequently, Yang and Hector proposed three new search models: *iterative deepening, directed BFS* and *local indices*. Each model is designated to meet certain aspects of metrics. The idea of iterative deepening is straightforward, where queries of iterative searches are flooded to successively larger numbers of peers, until either the query is satisfied, or the maximum depth limit has been reached. The intuition of the approach is to minimise the cost of query processing because the number of peers at each depth grows exponentially. The iterative deepening approach is suited to systems where satisfaction is the metric of choice. However, this is not applicable when the responding time is important [95]. The directed BFS technique implements *heuristic* selection strategies for selecting a subset of neighbours that are "believed" to return quality

results. However, the quality of a selection strategy is vital to the search performance as an inappropriate strategy can lead to insufficient results returned to the requesting node.

Finally, the local indices technique allows a peer maintains an index over the data of each peer within a radius of itself. The technique compensates the efficiency of query processing with the cost of space used for storing the data indices. Hence, every search needs to contact just a few peers, making the query routing more scalable. However, extra steps are required to create and maintain the indices at each node. The local indices technique is similar to the concepts of "super-nodes" in FastTrack [97] and the "ultra-peers" in the enhanced Gnutella [108]. Super-nodes are peers with the higher computing power and the networking capacity. They are responsible for caching the indices and for performing search queries on behalf of normal "user" nodes. In contrary to static servers, super-nodes are chosen from user nodes when they meet the performance criteria [97]. Nonetheless, the FastTrack protocol is the foundation of several popular P2P file-sharing systems including iMesh [98] and Kazaa [36].

Kalogeraki *et al.* [109] proposed a probabilistic BFS search, where a peer randomly selects a subset of its neighbours to forward query messages. The advantage of the probabilistic approach is that it requires fewer messages than the full flooding algorithm. So the probabilistic BFS can be more scalable than the Gnutella protocol. However, the approach is probabilistic and may result in an extremely long lookup period. Thus, they designed a profile mechanism that allows a peer keeping the most recent replies and a ranking mechanism that uses the profiles to find the most likely peers based on a query. On the other hand, the ranking technique can potentially cause query messages to be locked into a cycle as depicted in [109]. As a result, a peer needs

to select a small random subset of peers and add it to the set of best peers for each search.

The random nature of the probabilistic flooding can significantly reduce the traffic volume as well as the query coverage range. However, not all peers can be reachable using the probabilistic approach. It creates a so-called *partial coverage* problem [110]. Zhuang *et al.* [110] studied *Periodical* functions that are able to control the number of forwarding neighbours based on the TTL value. The flooding mechanism using a periodical function is termed *Periodic Flooding* (PF). In addition, two metrics – the quantity of shared data contained in a peer and the link latency – are incorporated with the periodic flooding in order to increase the hit rate and to reduce the lookup time.

Depth-First-Search (DFS) is an alternative search algorithm that has been previously studied. With DFS, each peer forwards the query to a single neighbour and waits for a response from the neighbour, before forwarding the query to another neighbour if the query was not satisfied. Similar to the Gnutella protocol, a variable embedded in a query message controls the depth of query routing to prevent the infinite lookup chains. Freenet [88, 93] is a DFS-based P2P system that facilitates storage and retrieval of distributed information. Each Freenet peer maintains its local *datastore* and a dynamic routing table containing addresses of other peers and the keys that they are thought to hold. The local *datastore* [88] is a shared directory that makes available to the network for reading and writing.

Query routing in the Freenet is similar to the style of IP routing, where queries are passed along from peer to peer in a chain of proxy requests. The next peer is selected based on the closeness of the request key and the keys kept by the dynamic routing

table. Each query is given a Hops-To-Live (HTL) limit that is decremented at each peer to prevent infinite chains. Loops incurred in lookup chains can also be avoided by assigning every quest a pseudo-unique random number such that peers can reject requests that have already seen before. Furthermore, the Freenet provides anonymity and it introduces a novel indexing scheme where a key to file is generated by hashing the content of the file.

Alternatively, a random walker model has been proposed in [96, 111, 112] where only one neighbour is chosen at each cycle of the query propagation. This significantly cuts down the total number of queries that is required. The weakness of the random walk model is that the expected search path is extremely long due to the randomness of the algorithm. A better approach is to use N random paths in the search [96, 112]. However, the choice of N has significant impact on the efficiency because use of excessive random walkers leads to the same problem in flooding.

Although random walkers can result in better utilisation of the P2P network than flooding, they have two basic problems. Firstly, the peer selection is unbiased and does not consider any indication of how likely it is that the peer will have responses for the query. Secondly, a query may possibly be queued in an overloaded peer for a long time before it is handled. Adamic *et al.* [113] addressed the first problem by using a biased random walker model. The search protocol of the model is biased towards high-degree peers. The intuition is that high-degree nodes, which they have the larger number of indices of files from neighbouring peers, will be more likely to have an answer that matches the query. The GIA P2P system [114] further improves the biased random walker model by considering the capacity constraints associated with each peer in the

P2P network. Therefore, queries are directed towards high-capacity nodes to achieve the better scaling and the load balancing.

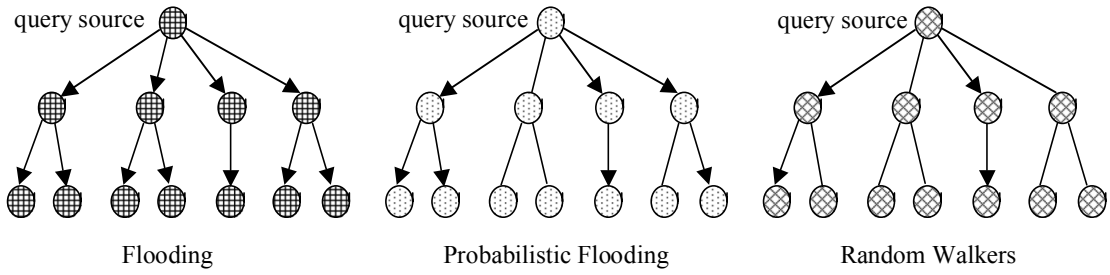

Figure 2: An illustration of three ad-hoc search algorithms: flooding, probabilistic flooding and *N* random walkers (*N*=2).

Figure 2 illustrates query lookup algorithms of flooding, probabilistic flooding and random walk respectively. Analytic work on the performance of three models has been studied in [111]. The study showed that the probabilistic flooding scheme and the random walker scheme have a better scaling property since they require less query messages on each search than the flooding-based protocol. Although unstructured systems generally do not guarantee that any request can be answered or the cost of an entity lookup is bounded, unstructured P2P systems are flexible for adapting complex queries and different performance metrics. Furthermore, several self-learning techniques [115, 116, 117, 118] have also been examined to tune the performance of query forwarding in unstructured peer-to-peer networks.

2.6.2. Static routing on structured overlay (SRSO)

In this category, the entity lookup relies on a *key* that is uniquely assigned to an entity. An entity key is a location-transparent name that is uniformly generated from a large space of identifiers. On the other hand, every peer is also given a single *identifier* chosen from the same identifier space. Subsequently, the entity keys are mapped to peers by an overlay network on the basis of node-content attributes, such as a

participating peer's IP address or stored data. Because the overlay networks are specifically structured and tightly controlled, they can change search from a standard graph-traversal problem into a *localised iterative process*. The lookup process is performed in an incremental manner. Each peer brings the query in which the lookup key is embedded closer to the identifier of the peer – responsible for the entity key – in the identifier space. This enables efficient entity searching and more importantly makes the query process deterministic. However, such a routing scheme shows a static behaviour as queries that travel between two fixed points in the overlay will fall on the same route unless the overlay is reconstructed.

A number of problems arise from SRSO due to the tightly controlled data placement and the inflexible overlay. The topology of the structured networks is generally independent to the organisation of the underlying physical network. This leads to the so-called *Topology Mismatch* problem [41, 42, 43]. As a result, the adjacent peers on an overlay network may reside distantly in the physical network. In addition, structured P2P networks incur significantly higher overheads than unstructured P2P networks in regard to the overlay maintenance. An updating mechanism is needed to rearrange the peer connections when peers arrive at or depart from the network. Thus, this limits the practical use of structured networks with highly transient nodes. Further, the tightly controlled data placement separates the node that keeps the identifier of an object from the node that owns the object. Although such a design improves both the load balancing and the resilience to node failures, it significantly increases the cost of communication.

Ratnasamy *et al.* [42] showed that non-uniform traffic can cause query hotspots and routing hotspots in structured models. Query hotspots are the result of some popular entities that are frequently being requested by other entities. This problem can be

resolved by caching the popular entities. On the other hand, routing hotspots are caused by unbalanced requests sent by some entities, leading to link congestions on the network. This issue is harder to overcome because of the use of fixed routing protocols in SRSO. Finally, the flat namespace used in structured models destroys entity locality [40]. That is, related entities may not be located closely in a structured overlay network.

Despite the differences of overlay topologies, structured P2P networks often support a distributed hash table (DHT) interface for providing a mapping function of a key to an authoritative peer, and share rather similar performance in the query forwarding. Commonly used topologies include Mesh [34, 38], Ring [23], d-dimension Torus [39] and *K*-ary tree [119]. A comparison of the performance of the existing structured P2P networks is given in Table 1. The key performance measures are summarised as follows.

- *Query forwarding cost* – it measures the number of peers in the overlay is needed to visit during the lookup of the querying entity;

- *Routing state* – it refers to the average size of a routing table embedded in a peer; and

- *Node insertion or deletion* – it shows the cost of re-structuring overlay connections when a peer joins or departs the network.

P2P Network	Architecture	Parameters	Routing cost	Routing state	Peer joins and leaves
Tapestry	Plaxton Mesh	N – number of peers in network B – base of the chosen identifier	$O(\log_B N)$	$\log_B N$	$\log_B N$
Pastry				$B \cdot \log_B N$	
Chord	Uni-dimensional ring	N – number of peers in network	$O(\log N)$	$\log N$	$(\log N)^2$
CAN	Multi-dimensional torus	N – number of peers in network d – number of dimensions	$O(d \cdot N^{1/d})$	$2 \cdot d$	$2 \cdot d$
Viceroy	Butterfly network	N – number of peers in network	$O(\log N)$	$\log N$	$\log N$
Kademlia	XOR metric for distance between points in the key space	N – number of peers in network B – based of the chosen identifier	$O(\log_B N)$	$B \cdot \log_B N + B$	$\log_B N$
k-ary search	*k*-ary tree	k – number of sub-trees under each node	$O(\log_k N)$	$(k - 1) \cdot \log_k N$	——

Table 1: A comparison of various structured Peer-to-Peer networks.

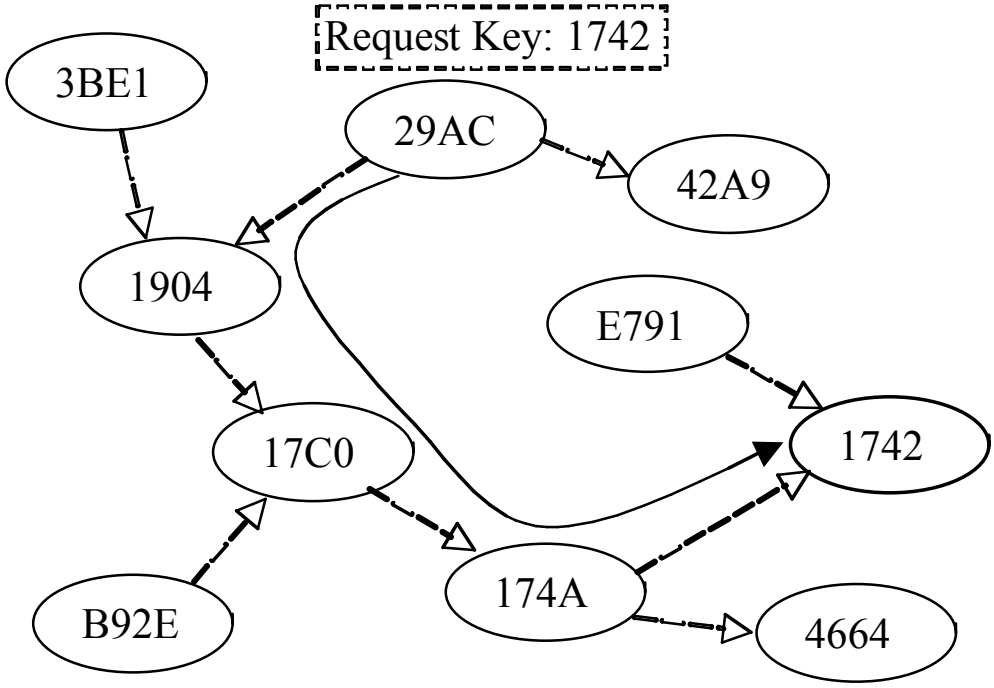

Figure 3: The travelled path on a Tapestry network by a query message with the requested Key *1742*. The query was initiated at Peer *29AC* and destined at Peer *1742*.

Pastry [38] and Tapestry [34, 120] are locating protocols based on Plaxton Mesh [60] and thus they share many similarities with each other. The Plaxton Mesh was originally developed for routing web queries to nearby caches. In the model, each node can take on the roles of servers (where entities are stored), routers (which forward queries) and clients (origins of queries). It implements a local routing table at every peer to incrementally forward query messages along peers whose node id is getting closer to the requested key. Subsequently, the query will arrive at the root peer – the authoritative node of the requested key. An example of a path of a query message is demonstrated in Figure 3. Such an algorithm is similar to the longest prefix routing in the CIDR IP address allocation architecture [121].

Every Plaxton peer maintains a multiple-levels routing table, where each level is corresponding to a digit position in the key space. To locate the next route, the n^{th} peer examines the $(n + 1)^{th}$ level map to locate the entry matching the value of the next digit in the requested key. The routing protocol can ensure that any existing unique key can be located by visiting at most $\log_B N$ peers, where N is the number of peers in the network and B is the base of the key space. To improve lookup efficiency, replicas of an entity are cached at a number of servers and pointers to those servers are stored at the paths leading to the root peer.

A similar approach is used in the lookup and routing mechanism of Tapestry. However, the Tapestry stores the locations of all data replicas to improve semantic flexibility and it allows an application-level to choose a replica based on some selection criteria. To resolve the problem of a single point of failure due to a single entity's root peer used in Plaxton, the Tapestry assigns multiple roots to each data entity through *surrogate routing* during the data publishing process to insert location information into the network. The surrogate routing enables to map any entity key uniquely to a live node whose node identifier has the longest prefix match. To detect link and server failures during normal operations, the Tapestry relies on TCP timeouts and UDP periodic heartbeats packets. Through utilising multiple root peers and maintaining periodically updated cached contents, the Tapestry can enhance the capacity to detect, to circumvent and to recover from system failures. Furthermore, the Tapestry is the fundamental component of the Decentralised Object Location and Routing (DOLR) API [120]. The DOLR API defines a set of programming interfaces for providing scalable location-independent routing. The DOLR API is designed for an Internet-scale, highly available storage utility termed the OceanStore system [89] and hence resiliency to node arriving and departing is a central focus in the API. Apart from the DOLR, other Tapestry-based

applications include the Bayeux [94] – an efficient self-organising multicast system and SpamWatch [122] – a decentralised spam-filtering system.

Pastry [38] also adopts the Plaxton-like prefix routing. However, it requires a larger routing state than the Tapestry. In addition to the routing table, every Pastry node maintains a *leaf set* that contains the node identifiers of a set of neighbouring nodes in the namespace. The leaf set provides the same functionality as the surrogate routing technique, where the destination can be reached quickly via the leaf set. A third routing state, termed neighbour set, is also maintained in every Pastry peer. While the neighbourhood set is not used for routing queries, it is constantly refreshed because the set plays an important role in exchanging information about nearby peers in order to maintain locality properties. The Pastry provides a foundation for a number of projects such as PAST [123] and Scribe [124].

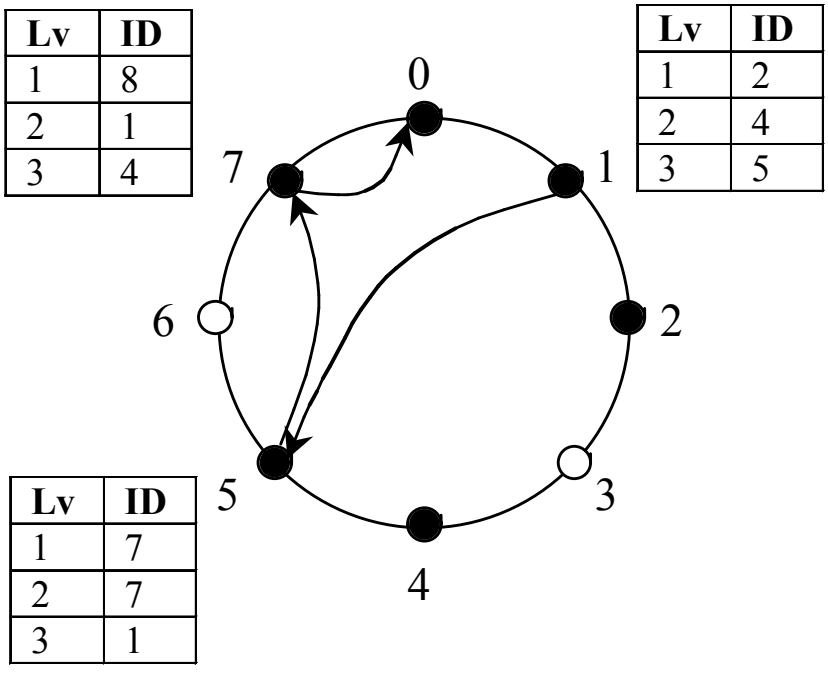

Figure 4: An example of a 3-bit Chord ring with 6 active peers as Peers **3** and **6** do not exist. A query path was established for the lookup of Key *0* originated at Peer ***1***.

Chord is a uni-dimensional structured P2P overlay network based on a ring structure. Compared to the Pastry and the Tapestry, the Chord has an advantage of simplicity in architecture. A Chord node has an *m*-bit identifier that is chosen by a *consistent hash* function [125] based on the node's IP address. Every identifier is location-independent and thus it is transparent to the structure of the physical network. The length of the identifier *m* must be large enough to make the probability of keys hashing to the same identifier negligible. On the other hand, every entity is assigned to an *m*-bit key that is generated by the same hash function. That is, the node identifiers and entity keys are located within the same identifier space. An entity key is stored in a node that has the identifier immediately followed. Such a node is known as the *successor* of the key [23]. A lookup process involves the matching of an entity key and a node identifier. Query messages are forwarded following the Chord circle clockwise via a series of successor pointers. The process halts when the messages arrive at the successor of the entity key. As long as the Chord ring can be maintained, every request is guaranteed to be deterministic.

To facilitate query routing, a Chord node maintains an *m*-level finger table, where the k^{th} finger points to a peer with the distance of at least 2^{k-1} on the overlay. The use of fingers is to accelerate the key search as the distance from the next forwarding node to the target node is at most half the distance from the current node to the target. Thus the number of forwarding nodes necessary will be $O(logN)$ under a stable condition [23]. An example of a 3-bit Chord is depicted in Figure 4, where peer **3** and **6** do not exist. Peer 1 invokes the *find_successor* function for the key **0**, which subsequently returns the successor of the key, i.e. peer **5**. Then, the query is forwarded to the next successor node along the ring, i.e. peer **7**, which is also the predecessor of the key **0**. Consequently, the

peer **7** passes the query to its successor – the authoritative node and subsequently the response is returned from the peer **0** along the reverse of the path afterwards.

It is essential that the successor pointers and fingers are up to date because the correctness of the lookup cannot be ensured or a query message needs to travel along an extremely long overlay path. The worst case is O(*N*), where a query has to visit every peer on the ring when the fingers are out-of-date. The Chord protocol uses a *stabilisation* protocol running periodically in the background to refresh fingers within the routing table. El-Ansary *et al.* [119] observed that the Chord lookup algorithm mimics binary search and subsequently, a generalised algorithm based on a *k*-ary tree search has been given in [119]. The *k*-ary tree search reduces the length of the search path to O($log_k N$), while the number of items stored in the routing table in each node increases to O($(k-1)*log_k N$).

The simplicity of Chord comes with a price. Firstly, the design of the Chord routing table is lacking of inflexibility as a finger in a peer's routing table always points to the same distant peer when the network is steady. This can cause some peers process more queries than other peers if the network traffic is not uniformly distributed. This problem is also known as *routing hotspots* [42]. Only a single peer is chosen at each level of a finger table in the Chord. This also raises concerns of the liability to peer failures and the resiliency to link congestions. Moreover, there is no natural correlation in the Chord between the namespace distance on the overlay and the proximity distance on the underlying network. As a result, query messages are routed regardless the distance traversed in the physical network.

Content Addressable Network (CAN) [39] is another structured P2P network. The notable feature of the CAN is that every node has a constant number of neighbours. That is, the routing state does not grow with the number of nodes. This can effectively reduce the complexity of an updating mechanism. The CAN maps nodes onto a *d*-dimensional Cartesian coordination space on top of a *d*-torus. Consequently, the coordination space is partitioned into a number of distinct zones and every peer possesses its individual zone. CAN query messages are routed towards the destination peer by comparing the coordinate included in these messages with neighbours' coordinates. In summary, the CAN's geometry produces an expected routing path with $O(dN^{1/d})$, with each node maintaining routing state for $2d$ neighbours. When a new peer joins the system, it will split an existing peer's zone into half; retaining half for the peer and allocating the other half to the new peer. The load balancing property is maintained by using a bootstrap process that supplies to a new peer with the IP addresses of some randomly chosen peers in the system. The new peer subsequently sends a JOIN message randomly to one of those peers generated from the bootstrap process. Improvement to the CAN routing algorithm can be achieved by adopting multiple, independent coordinate spaces, whereby each peer in the system being assigned a different zone in each coordinate space, termed *reality*.

The Viceroy [126] P2P overlay network maintains a Butterfly topology that uses links between successors and predecessors on the ring. The ideas of the Viceroy were inspired by early works of Kleingberg [107] and Barrière *et al.* [127]. The advantages of the Viceroy include its constant degree of links, similar to CAN, and logarithmic diameter of the overlay, similar to Chord. Therefore, the routing path of the Viceroy is achieved in $O(logN)$ on average and peers arriving and departing require only O(1) peers to change their states. However, Li and Plaxton [128] questioned the practical use

of the limited degree network as it may restrict the adoption of a proximate neighbour selection.

The Kademlia [129] resembles a P2P distributed hash table that maps each peer's identifier in a 160-bit key space. The pairs of (*key*, *value*) are stored on nodes with IDs close to the key for some notion of closeness. A novel XOR metric is used to measure distance between points in the key space. XOR is symmetric, allowing Kademlia peers to receive lookup queries from the same distribution of nodes included in their routing tables. The Kademlia uses a single routing algorithm that resembles the Pastry's first phase; a query message travels to a point, which is calculated by the Kademlia metric, about half of the remaining distance to the target ID at each step. Because the XOR metric is uni-directional like the Chord, it ensures that all lookups for the same key converge along the same path. As a result, in contrast to the Pastry and the Tapestry, the Kademlia does not require a secondary routing table to discover the target peer from a group of peers sharing with the same prefix up to (b - 1) digits.

2.6.3. Informed routing on constrained overlay (IRCO)

In this category, the overlay networks illustrate predictable topological properties and scaled query routing by putting constraints on the construction of routing tables and applying statistical and/or heuristic information to guard query lookups. Although some lookup protocols of unstructured networks have also explored probabilistic methods, they cannot locate rare entities efficiently. In contrast, query routing in constrained systems is based on a more systematically designed protocol that can find entities in an unbiased manner. The scalable characteristic of constrained networks thereby can be comparable to that of structured networks. Because constrained networks do not require assigning data with globally unique identifiers or placing data at specific peers, they can generally support both key-based or keywords search like unstructured networks.

Consequently, constrained overlay networks bring together some of the key features of two worlds: a flexible architecture and efficient entity location. Examples of these networks include mOverlay [100], Sandberg network [101], Scalable Query Routing (SQR) [102], YAPPERS [99], and Percolation Search [103].

The mOverlay P2P network aims to resolve the topology mismatch problem stemming from a randomly generated overlay network. Thus, the model takes into account of locality of network peers when constructing an overlay structure. The mOverlay network uses a simple mechanism to achieve the goal; peers of the network are organised into groups. Information is completely shared among all the peers within a group. The decision of which group a new peer should belong to is based on the grouping criterion; when the distance between a new peer and group *A*'s neighbour groups is the same as the distance between group *A* and group *A*'s neighbour groups, then the peer is assigned to group *A*. Here, the neighbour groups are acting as *dynamic landmarks*. To obtain these dynamic landmarks, however, there is a need for a global cache system called the rendezvous point (RP), which it returns the start points in the overlay, also known as *boot hosts*, to a new peer. Then the new peer needs to iteratively contact these boot hosts to determine its group. Because such a join procedure can take a long time, a stop criterion is essential to use.

The problem of finding a specific peer is similar to find a nearest group in the overlay for a new peer. Query routing can take advantage of the small network diameter due to grouping of peers. Given that each group has *M* neighbour groups, the average distance between any two nodes is $\log_M N$ with *N* peers in a network. Furthermore, because information is shared among peers in a group it needs to be updated regularly. This is done though flooding. Zhang *et al.* argued that the impact of the broadcast overhead

could be neglected as peers within a group are physically closed to each other [100]. On the other hand, information on neighbours is updated through leaders of groups. These group leaders function similarly to the FastTrack super-peers or the Gnutella ultra-nodes. Therefore, it is expected that these leaders can experience higher loads than other peers do.

On the other hand, Sandberg explored different clustered networks based on so-called small-world networks [101]. Small-world networks are originally observed within the social network by Milgram in 1967 [130]. However, it was not until Kleinberg's recent work [107] that a mathematical model for efficient routing in such networks was created. Kleinberg illustrated that the possibility of efficient routing depends on a balance between the ratios of short links (nearby contacts) and long links (long range contacts) established in a peer's routing table. Under a particular distribution, simple greedy routing can find routes in $O(\log^2 N)$ steps on average. However, this assumes that each node is aware of its own coordinates, its neighbours and the destination node. Subsequently, Sandberg designed a statistical approach that assigns positions to the nodes according to the a-posteriori distribution of the links. Here, these links are assigned according to Kleinberg's model as described earlier. A query path can be formed based on statistical information as analogised to the construction of a Markov chain. Here, this is performed by using the Metropolis-Hastings algorithm. The concept of small world networks has also been applied to randomised overlay networks. However, randomised networks are substantially different to the Sandberg model because they have to rely on a structured topology to provide coordinates of peers.

Kumar *et al.* [102] proposed a query routing protocol, termed Scalable Query Routing (SQR) that allows low bandwidth consumption during query forwarding. The protocol

makes use of a highly compressed, probabilistic-based routing table that provides a low cost mechanism to create and maintain information about nearby peers. A novel data structure – the Exponential Decay Bloom Filter (EDBF) was designed for creating such a routing table. Information of a SQR system is encrypted into a string of bits like a key. An owner peer publishes its information to its neighbours through propagation. However, the amount of information decreases exponentially with the distance from the hosting node. Consequently, peers close to the origin have strong information while peers resided far away from the origin receive indistinguishable "noises". In other words, an EDBF routing table is an information filter that varies the number of bits used to store probabilistic information. On the name lookup, each query follows a random walk until it hits a node within the aware neighbourhood. The aware neighbourhood is a group of peers that holds valid information about the request entity in its routing table. Next, the query is guarded by the amount of information (i.e. the number of match bits) stored in the EDBF table of each peer inside the aware neighbourhood and thus the query can quickly gravitate towards the owner peer. By following the corresponding routing table entries, a query message can reach the destination using a small number of hops with high probability. Because the accuracy of information stored in EDBF table has significant impact on the effectiveness of query routing. As a result, the SQR's routing tables need to be refreshed periodically.

Percolation Search [103] is another constrained P2P network that makes use of content publication and probabilistic broadcast in entity lookups as similar to the SQR. However, the model has been designed for a particular class of P2P random networks – the Power-Law (PL) networks that have heavy-tailed degree distributions. Taking advantage of the network characteristics, the percolation search algorithm is capable of finding any content in the network with probability one in time O(*logN*), with a total

traffic that scales sub-linearly with the network size N. There are three important steps defined in the percolation search algorithm, namely, content list implantation, query implantation and bond percolation. Every node needs to duplicate its content list through a random walk of size $L(N, \tau)$ initiating from itself. The exact form of $L(N, \tau)$ depends on the topology of the network. Sarchar *et al.* [103] have shown that such a random walk will visit at least one high-degree peer (i.e. high density of neighbours) with probability approaching one. In other words, a pointer to any content is owned by at least one highly connected peer. Before starting a query, a request message is implanted through another random walk of the same size $L(N, \tau)$ starting from the requester. Again, the random walk will pass through at least a highly connected peer. Subsequently, every node with a request implantation starts a probabilistic broadcast search. As the diameter of highly connected peers in a random graph is on the order of $O(logN)$, the expected search step will therefore be $O(logN)$. While the percolation search protocol shows a good scaling property, it can easily generate query hotspots as the algorithm largely relies on high degree peers. However, it is not clear how well the percolation search protocol manages transient peers or network failures.

YAPPERS (Yet Another Peer-to-PEeR System) is an interesting P2P network that operates on top of an arbitrary overlay network while providing the DHT-like search functionality. Similar to the mOverlay and the Sandberg, peers of the YAPPERS are grouped based on a hash function, e.g. one could hash the IP address of a node. Correspondingly, the key space of all the keys is partitioned into a small number of buckets based on the same hash function. When a peer needs to register a data entity for a key, it has to place the <*key*, *value*> pair in a peer whose group number coincides with the key's bucket number. For instance, if the key space is divided into two colours: white or grey, as well as the peers of the network, a white peer A is able to store any

white key at itself. On the other hand, peer *A* needs to search for a grey peer within *h* hops if it wants to register another data entity with a grey key. Here, the peers within *h* hops are called immediate neighbours of peer *A*. The basic idea of the YAPPERS routing algorithm is to forward a query to *all* peers of the same colour and not to disturb any peers that could not have answers. Consequently, such an algorithm requires far fewer query messages in lookup than the full flooding approach. In order to support a *Total Lookup* (i.e. all the same coloured peers must be visited for a lookup), the routing algorithm, however, requires each peer to maintain knowledge about its neighbours' neighbours. In general, the YAPPERS behaves like a DHT within an immediate neighbourhood while it uses a flooding approach to navigate between peers of the same colour outside the immediate neighbourhood, acting like an unstructured network. Furthermore, the YAPPERS resolves the query hotspot issue by processing the same request simultaneously on multiple peers. However, it may generate routing hotspots for locating popular entities due to the use of the flooding-based routing.

The ability of providing scaled query routing while maintaining overlay flexibility is the main advantage of constrained overlay networks. Compared to unstructured counterparts, constrained overlay networks induce a loosely structured topology, where the construction of routing tables is guarded by preliminarily defined rules. Clustering has been commonly used to build such a loosely structured topology. Peers are organised into a number of overlapping groups so leading to a small network diameter. Peers within a group are aware of contents of one another. Consequently, query routing can make use of groups and shared information to skip many peers that belong to the same group. In general, constrained systems can be distinct by their clustering strategies: 1) the mOverlay and the SQR use the proximity property of peers; 2) the YAPPERS relies on a hashing function like Chord; and 3) the Sandberg network and

the Percolation Search exploit the statistical characteristics of power-law networks. In contrast to structured P2P systems, constrained networks do not impose any regular topology on top of a P2P network. This enables constrained networks to adapt to the changing communication patterns and hence can improve the routing efficiency and the system robustness. On the other hand, it does not guarantee that every request can be resolved through such an irregular topology. Further, all abovementioned systems require periodically flooding peers within a group in order to refresh their content indexing table. Such periodic flooding may have a major impact on the loads of communication links.

2.6.4. Proximate routing on randomised overlay (PRRO)

The need for incorporation with proximate neighbour and routing selection in structured P2P systems has led to development of randomised overlay models [105]. Similar to a constrained overlay, a randomised model aims to offer benefits from both structured networks and unstructured networks. However, the two models approach it differently. The randomised overlay is constructed by imposing a random graph to a structured geometry such as hypercubes. The important characteristic of the randomised overlay is its degree of flexibility [43], which refers to the amount of freedom available to choose neighbours and next-peer paths. Consequently, the flexible structure is able to achieve better resilience and proximity performance than the rigid architecture of the structured systems.

Query routing can be benefited from the flexible yet consistent randomised structure. The base graph of a randomised structure exploits a regular geometry, which it can guarantee that every request is resolved and has sound routing performance like a structured network. On the other hand, these randomised structures can route queries via multiple paths to reduce hotspots in the network. The major drawback of randomised

networks, however, is that the routing state is generally bigger than the structured networks. This may have impact on the load balancing of the P2P overlay.

Randomised P2P systems can be classified based on the proximity algorithms [131]: proximate neighbour selection and proximate routing selection. The idea of the proximate routing selection is to select, among the possible next peers, the one that is closest in the physical network or the one that represents a good compromise between progress in the id space and proximity [41]. Hence, with *k* alternative peers in each step, the approach can reduce the overall delay in query forwarding between the source and the destination. Systems of this group include AntCAN [105], Symphony [35], NoN [132] and RASTER [106]. Alternatively, the proximate neighbour selection constructs a topology-aware overlay by choosing routing table entries to refer to the topologically nearest among all nodes with node id in the desired portion of the id space [41]. The key difference between the two selection algorithms is that the proximate routing selection does not alter the network topology, or the routing tables. Randomised systems centred at the proximate neighbour selection are randomised-Pastry [41], randomised-Chord [104] and SkipNet [133].

AntCAN described by Apel and Buchmann in [105] is a variant of the CAN. It adapts an Ant Colony Optimisation (ACO) algorithm to improve the query processing in a highly dynamic environment and evolving requirements. The AntCAN allows a peer to keep a limited number of other contacts in addition to its neighbours. Contacts provide shortcuts to some peers and are populated by information contained in rely messages. However, the contacts are not organised structurally. The ACO algorithm is thereby used to optimise the size of contacts. In general, ACO is primarily used to refresh the contacts of the AntCAN, which is similar to a cache management.

Symphony [35] and NoN (Neighbour-of-Neighbour) [132] are both inspired by the Kleinberg's work of small-world percolation networks [107]. As mentioned previously, the notable feature of those networks is the small network diameter and thus a routing algorithm is able to diminish route lengths to O(log*N*/loglog*N*) peers. While the Symphony illustrates a construction of a small-world percolation network on the basis of ring geometry, the NoN provides a general technique to optimise the query forwarding performance for any structured networks. The idea behind NoN is to consider a neighbour's neighbours for making better routing decisions. However, the approach requires a substantial amount of spaces to keep the information of neighbours' neighbours.

Wang *et al.* [106] thereby proposed a bitmap-encoding scheme to encode and aggregate routing information. The result of the information aggregation is a compressed/encoded representation of a set of nodes in a structured network, termed *bitmap*. Formally, a bitmap of resolution *r* and covering a radius *a* from an arbitrary node *s*, $B_s^{a,r}$, is an *r*-bit binary string that represents the overlay positions that are reachable within *a* hops from *s*. The system's logic space is hence evenly partitioned into *r* equal-size virtual bins matched to the bits of the binary string. Further, a bit is set to one if there exists at least one peer in corresponding bin that is reachable within *a* hops from *s*. A novel routing protocol, termed RASTER, can make use of bitmaps to approximate the shortest overlay route between peers through straightforward bit-wise operations. When RASTER cannot find any matching bins in a bitmap, it will forward the query via the default routing scheme of the base structured system. Like other proximate routing algorithms, the RASTER protocol can only reduce the expected per-peer delay to the smallest expected delay amongst a slight number of nodes with random locations in the

network. Consequently, proximate neighbour selection is a prior solution to the topology mismatch problem.

Randomised-Pastry [41] and randomised-Chord [104] perform a routing table maintenance task to sustain the locality properties of the routing table over time. With the randomised-Pastry, the task is performed on a periodic basis, where a peer selects a random entry in the row, and requests from the associated peer a copy of that peer's corresponding routing table row. Subsequently, each entry in that row is compared to the corresponding entry in the local routing table. If they are different, the peer probes the distance to both entries and installs the closest entry in its own routing table. On the other hand, the randomised-Chord proposed an incremental approach to improve lookup latency. When a request is completed, every peer on the search path samples the target, and updates its finger table accordingly. Such a technique is termed Lookup-Parasitic Random Sampling (LPRS) since all information that the random sampling scheme needs is contained in the path that a lookup request traverses [104]. Nonetheless, the randomised-Chord requires relaxing the original construction of fingers tables in the Chord [23]. It allows the k^{th} finger to point to any peer whose node identifier is in the range of [k^{th}, $(k+1)^{th}$] without affecting the expected path-length.

SkipNet [133] is a new overlay based on the skip list [134] – a randomised balanced tree data structure is organised as a tower of increasing sparse linked lists. The feature of the SkipNet is a scalable overlay network that provides support for path locality properties as well as the DHT functionality. The path locality means that the traffic between two overlay peers within the same domain traverses within that domain only. The path locality not only benefits topology-aware routing but also provides additional security to protect contents within a domain. In spite of this, the SkipNet has not

considered any adaptive mechanism to maintain the locality properties of the routing tables. This may limit the performance of the SkipNet due to the dynamic nature of peer-to-peer networks.

To conclude this chapter, characteristics of previously discussed P2P overlay networks are summarised in Figure 5. Chapter Three will introduce a novel decentralised naming scheme directed at object-based distributed computing structures.

Figure 5: A summary of Peer-to-Peer (P2P) overlay networks.

3. A Novel Decentralised Naming Scheme Directed at Object-based Computing Structures

3.1. Naming Supports for Message Passing and Object Migration

Message passing between objects in a distributed object system relies on a name to differentiate the recipient of the message. A dynamic naming system, which manages bindings and resolutions of all symbolic names, can have a significant impact on the efficiency of message passing. To achieve good performance in communication, the naming system must facilitate the patterns of message passing as characterised by object-oriented software running on the system [47]. The analysis of communication patterns in object-oriented software is essential to this research and hence Chapter 4 considers this particular issue. The analysis revealed a number of communication patterns in selected object-oriented benchmark programs.

- Message creations, which result in new objects being created in the system, were frequent and clustered. This implies that a naming system needs to concurrently bind names to objects to improve the rate of the object creations.

- Non-uniformed distribution of the number of messages exchanged between objects; the phenomenon that a small group of objects send or receive significantly more messages than the majority of the other objects.

- Message workloads (in terms of messages being sent or received) of an object tended to correlate with its lifespan. That is, short-lived objects had smaller message workloads.

- Message passing exhibited the locality of reference (*temporal* and *spatial*) phenomenon. The locality properties are attractive for designing strategies to optimise the name lookup.

- The dynamic object creation and the dynamic object acquaintance via names in messages imply that the topology of object relationships potentially varies through the execution of object-oriented software. The changing topology can potentially alter the pattern of message passing.

Apart from the patterns of object communication, migration support for distributed object systems has also been considered in this study since object migration is a key to resource management. Object migration enables dynamic load distribution, fault resilience, and data access locality. It is desired that a naming system can resolve the new address of the migrated object without changing the object's original name or affecting the application software at all.

ARMS is a novel decentralised naming model specially targeted at object-based distributed computing systems. It supports message passing via an ***A****daptive* locating protocol, a ***R****andomised* overlay network and a ***M****igration-enabled* relocation ***S****cheme*. The characteristics of ARMS include

- *Support for concurrent name bindings and resolutions* – the heart of ARMS is a peer-based naming model that enables names to be bound and resolved locally since the name space is distributed across nodes in the network. This essentially facilitates parallel object creation and name resolution;

- *Provision of a flexible overlay topology* – a randomised overlay is adopted to address the hotspot and topology mismatch issues. As mentioned earlier, some objects received more messages than others did. This would cause some nodes in the decentralised naming system received more name queries, and potentially create hotspots in the system. The randomised overlay allows redundant logical paths to be established between nodes as similar to an unstructured overlay. These redundant paths can help to reduce hotspots in the non-uniformly name queries. Besides, the topology of the overlay can be dynamically changed to be compatible with the structure of the underlying communication network. Consequently, this can increase the efficiency of query forwarding;

- *Efficient query routing based on path exploitation and exploration* – the query forwarding in ARMS is controlled by an adaptive locating protocol that integrates a distributed hash table (DHT) with the ant colony optimisation (ACO) algorithm [135]. The ACO algorithm can take advantage of redundant paths in the randomised overlay to explore the best path to route the name query, based on a set of constructive heuristics, in face of the dynamic communication patterns or network conditions. Further, the algorithm removes slow links from the routing table and maintains the table size with only the good links;

- *Enabling object migration* – ARMS has implemented a hybrid relocation scheme to support transparent object migration. The hybrid scheme combines three common relocation techniques: *home-based*, *forward pointers* and *backtracking*. The relocation protocol is an extension of the adaptive routing protocol and hence two protocols can be swapped smoothly; and

- *Message routing with location transparent names* – ARMS allows a distributed object system to route messages to destinations based symbolic names, rather than their physical addresses. This approach can eliminate the interruption by performing the name resolution with the server-based naming system. However, the route that is selected based on the name-based routing tends to be different to the route based on the address-based routing because of the discrepancy between the overlay topology and the physical network. The use of the adaptive routing protocol and the randomised overlay is important to close the gap of two types of routing.

3.2. Placement of a Random Graph over the Chord Topology

ARMS overlay network is adapted from the Chord [23]. The Chord is a popular structured network and implements a DHT-based locating protocol via a ring topology. In the Chord, each node has an *m*-bit identifier that is chosen by a consistent hash function [23] based on the node's IP address. Such an identifier is location-independent and thus it is independent to the organisation of the physical network. On the other hand, every object is assigned to an *m*-bit identifier that is generated by the same hash function. That is, the node identifiers and object identifiers are located on the same identifier space. The importance of the hash function is its local assignment of the object identifiers. Such a function can eliminate a centralised approach for binding objects and thus support concurrent object creations in the system.

Every node is responsible to only a subset of identifiers on the ring space in order to balance the state of the routing table. The identifier of an object is stored in a node that has the identifier immediately followed. Such a node is known as the *successor* of the

identifier. For instance, node **3** is the successor of Identifier **63** and node **6** is the successor of Identifier **5** as illustrated in Figure 6.

For *m* bits identifiers, each Chord node maintains an *m*-level finger table. The finger table stores a pointer that is pointing to the successor of the key:

$$(n+2^{k-1}) \bmod 2^m,\ 1 \le k \le m \tag{3.1}$$

where *n* is the node identifier, *k* is the k^{th} level of the finger table. The use of fingers is to accelerate the key search as the distance from the next forwarding node to the target node is at most half the distance from the current node to the target. Thus the number of the forwarding nodes necessary will be $O(\log N)$ under a stable condition [23].

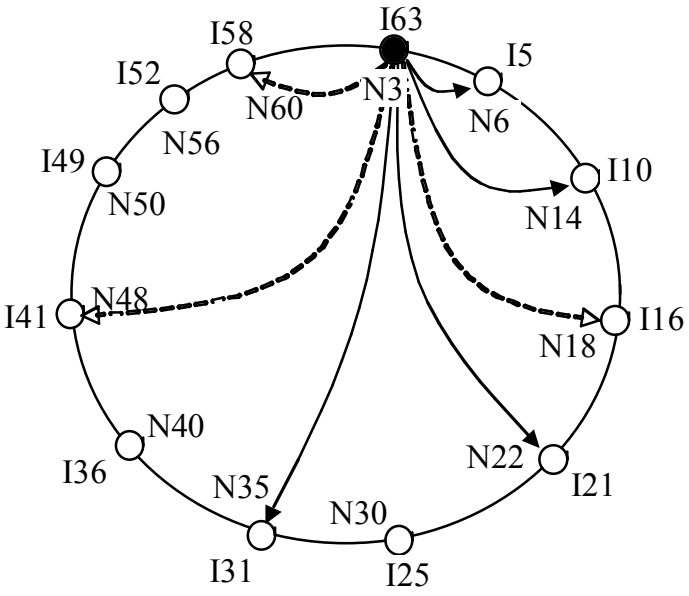

Figure 6: An illustration of the Chord ring architecture.

While only one node is allowed to store its address at each level of the Chord's forwarding table, ARMS allows multiple addresses to be inserted into the k^{th} level of the table for nodes whose identifiers are belonged to the range of $(n+2^{k-1})$ mod 2^m and $(n+2^k)$ mod 2^m. Although this design breaks the load balancing of the Chord's forwarding table, the flexibility of the neighbour selection can improve the routing efficiency and latency resiliency. Because a peer can select more than one route at each lookup, it can enhance the routing efficiency by selecting proximate peers on the physical network. Furthermore, multiple links can effectively reduce hotspots by shedding loads amongst redundant paths. ARMS supports the dynamical arrangement of the finger table to tune the performance of the communication system according to the traffic pattern.

Another significant difference between the two models is that the identifiers of both objects and nodes are generated randomly in the Chord – so the owner of an object may not be the same node that owns the object identifier – while only the node identifiers are randomly generated in ARMS – the object identifiers are assigned by the nodes that own the objects. The motivation of ARMS is to allow transporting a message directly based on an object's identifier to the owner of the object. This arguably trades increasing communication efficiency for higher load balancing cost.

An example of an ARMS forwarding table is illustrated in Table 2. The first column lists the k^{th} level of the table; the second column gives the values of the key identifiers calculated by Equation 3.1; the third column gives the range of the key identifiers for that level; the next column gives the successor of the key for each level; the last column gives the number of nodes corresponding to the level. The use of the table for query forwarding is presented in the following section.

Level	Start	Interval	Successor	Node(s)
1	4	[4-5)	N6	N6
2	5	[5-7)	N6	N6
3	7	[7-11)	N14	N14
4	11	[11-19)	N14	N14, N18
5	19	[19-35)	N22	N22
6	35	[35-2)	N35	N35, N48, N60

Table 2: A snapshot of the forwarding table at node N3 in ARMS.

3.3. Adaptive Locating Protocol

3.3.1. Ant-inspired routing

The adaptive locating scheme in ARMS integrates a distributed hash table (DHT) protocol with a distributed Ant Colony Optimisation (ACO) algorithm. ACO is a probabilistic technique for solving optimisation problems, which can be abstracted to find the best path on a graph [135, 136]. The ACO algorithm is inspired by the behaviour of some ant species in searching paths from the colony to the food source. In analogy to biology ants, artificial ants deposit a substance termed pheromone on the trail in order to encourage other members to follow. The importance of the pheromone is to stop ants wondering at random but to instead utilise and to reinforce the path that eventually leads to the food source. Over time, the pheromones start to evaporate and thus the trail loses its concentrations. The more frequently ants travel down the trail, the higher concentrations the trail can have and so the longer time the process of the pheromone evaporation takes. Through this, the pheromones of the good or promising trails will be strengthened and the bad ones will be weakened.

In ARMS, query messages are analogous to ants. They select a path via a stochastic mechanism that is based on the values of the pheromone and the heuristic information. Here, the pheromone is used to exploit the good solutions that are discovered previously. The use of the heuristic information is twofold. First, the constructive

heuristic can provide guidance to messages at the early stage of the search when the pheromones are not developed yet. Secondly, it helps a message to choose an alternative path in case a previously known solution becomes an obstacle due to the change of the network conditions. Such a mechanism is vital for support path exploration in a dynamic environment. In addition, parallel messages are applied initially in order to build a route quickly. This is similar to the *N*-walkers approach [96, 112]. Once the target object is found, the pheromones on the fastest path are updated via a reply message. This helps to improve the efficiency of future searches for the same target by other objects. Because searches from different querying objects will converge towards the target on the randomised structure eventually, the strong pheromones can increase the convergent rate of those queries. The profiling study in Chapter 4 showed that nearly 50% of remote invocation calls required returning values. Therefore, the updating approach should not dramatically increase the network traffic in the system since updating information can be piggybacked on the returned message.

On the other hand, the pheromones on a path that is not updated will decay over time. Generally, the decomposition of pheromones progresses faster on a longer path compared to a shorter one. By setting a threshold value, inefficient links – the pheromone concentration drops below the threshold – will be removed from the randomised network. Thus, the topology of ARMS network is not fixed but constantly evolving like an unstructured network. This is vital to optimise the size of the routing state of a peer and thus the overall number of links in the overlay network can be minimised.

3.3.2. Algorithm description

Figure 7 depicts the algorithm of object locating in ARMS. Upon the object locating, the source node first checks whether the identifier of the request object lies at its

successor. If it is found, the query is forwarded to the successor and then the locating process is terminated. Otherwise, the forwarding table is used to propagate the query message. The source of the query message will forward a q number of parallel messages to q neighbouring nodes. The intermediate nodes will subsequently propagate the query to just one of its neighbours. In contrast to finding the "closest" node on the key space, which it is required in most structured networks, ARMS node looks for one peer amongst a set of promising neighbouring peers. This is determined by a probability process [135]:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}, \textbf{ if } C_{ij} \in N(S^p) \tag{3.2}$$

where τ_{ij} is the pheromone associated with the edge joining node i and j, $N(S^P)$ is the set of unvisited neighbours, the parameters α and β control the relative importance of the pheromone and the heuristic information η_{ij} which is given by:

$$\eta_{ij} = \frac{1}{d_{ij}}$$

where d_{ij} is the distance of identifiers of the node i and j on the ring space. Other choices of the heuristic information are discussed later in the chapter.

PROOF. Suppose A, B, C be separate nodes on the ring such that $A.id < B.id < C.id$. Particularly, node A is the source of the query and node C is the destination while node B is an intermediate node on the path to node C.

Further, it is given that node C is resided at the i^{th} level in the forwarding table of node A, such that $C.id \in (2^{i-1}, 2^i)$ and $1 \leq i \leq m$, where m is the number of bits in the node identifier. Suppose a function D_{XY} to return the logic distance between node X and Y on the virtual structure. It is important to prove $D_{BC} \leq h \cdot 2^i$, where $h < 1$ and i is the i^{th} level in the table; $1 \leq i \leq m$.

```
peer.locate(key)
    if key ∈ (this.id, successor.id)
        forward the query to successor // Target found
    else
        Let C be an empty set
    end-if

    for i = m downto 1
        for all nodes n' in finger[i]
            if n'.id ∈ (this.id, key)
                add n' to C
            end-if
        end-for
    end-for

    return C
end locate
```

```
peer.send_queries(C)
    if this is the source node
        choose N nodes from C with probability p_ij given by Equation 3.2
        send N queries to N nodes
    else
        choose one node from C with probability p_ij given by Equation 3.2
        send a query to that node
    end-if
```

end send_queries

Figure 7: The object location protocol defined in ARMS.

Theorem 1. With a stable structure, the number of nodes, k, that must be contacted to find a successor in an N-node network is O(log*N*).

Case 1: B.id, C.id $\in (2^{i-1}, 2^i)$

Suppose that there is an intermediate node *B*, where $B.id \in (2^{i-1}, 2^i)$ and $2^{i-1} \leq B.id \leq C.id$. Hence, the distance between node *A* and node *C* is given by:

$$D_{AC} = D_{AB} + D_{BC} \leq 2^i \tag{3.3}$$

Because node *B* and node *C* are both located at the i^{th} level of the forward table in node *A*, the distance between node *A* and node *B* is at least 2^{i-1}:

$$D_{AB} \geq 2^{i-1} \tag{3.4}$$

In turn, substituting Equation 3.4 into Equation 3.3 gives:

$$D_{BC} \leq 2^i - 2^{i-1}$$

$$D_{BC} \leq 2^{i-1} = \frac{2^i}{2} = h \cdot 2^i, \text{ where } h = \frac{1}{2}$$

As the intermediate node *B* is selected in a descending order from the forwarding table, i.e. $\{i: m \rightarrow 1\}$, the distance D_{BC} will be halved at every step. Therefore, the distance

D_{BC} will become one (i.e. node *B* and node *C* are immediate neighbours) after $k = m$ steps.

Since node identifiers are randomly generated, the number of identifiers is evenly distributed among nodes and it is given by [23]:

$$\frac{2^m}{\sigma} = N \Rightarrow m = \log_2 N + \log_2 \sigma \tag{3.5}$$

N is the total number of nodes. Substituting $k = m$ into Equation 3.5,

$$k = \log_2 N + \log_2 \sigma$$

Therefore, the average number of the forwarding steps, *k*, will be $O(\log N)$ with a steady network topology.

Case 2: $B.id \in (2^{i-2}, 2^{i-1})$, $C.id \in (2^{i-1}, 2^i)$

As the table size varies at runtime, it is possible that there is no such node B' satisfies the condition $2^{i-1} \leq B'.id \leq C.id$. Suppose that each level of the table keeps at least one entry, there will be a node *B* at the $(i\text{-}2)^{\text{th}}$ level, where $B.id \in (2^{i-2}, 2^{i-1})$. The distance between node *B* and node *C* is therefore given by:

$$D_{BC} \leq 2^i - 2^{i-2}$$

$$D_{BC} \leq \frac{3 \cdot 2^i}{4} = h_1 \cdot 2^i, \text{ where } h_1 = \frac{3}{2} \cdot h$$

Again, when *i* descends from $m \rightarrow 1$, the number of steps is given by:

$$k = h_1 \log_2 N + h_1 \log_2 \sigma$$

Therefore, the number of forwarding steps, k, will be $O(\log N)$ as proved.

3.3.3. Updating mechanism

With ACO, the values of the pheromones are updated after a solution is established. Then, the solution is refined in the following iterations until the number of iterations excesses the preset limit. However, the update mechanism has to be modified to fit the distributed environment.

The update mechanism of ARMS is closely related to the MAX-MIN Ant System (MMAS) [137]. In MMAS, only the best ant updates the pheromone trails and the pheromone value is bound to a specific range. Consequently, when the requested object is found in the distributed system, a reply message is returned from the destination site via the fastest forwarding path. However, due to the distributed nature of ARMS, it is impossible to perform iterative updating for each link's pheromones. To resolve the problem, the values of pheromones are decaying according to time. Subsequently, every node on that path updates its pheromone:

$$\tau = \begin{cases} [e^{-((T/T_{LUT})\bullet\rho)} \cdot \tau + \Delta\tau]_{\tau\min}^{\tau\max} & \text{if it is on the best path} \\ [e^{-((T/T_{LUT})\bullet\rho)} \cdot \tau]_{\tau\min}^{\tau\max} & \text{; otherwise} \end{cases} \quad (3.6)$$

where ρ is the evaporation rate, T and T_{LUT} are respectively the current time and the last update time of the pheromone value, τ_{max} and τ_{min} are the upper and lower bounds imposed on the pheromone and $\Delta\tau$ is defined as:

$$\Delta\tau = \frac{Q_H}{L}$$

where Q_H is a constant, L is the length of the path.

Furthermore, to encourage to select different neighbours for query forwarding, the pheromone of the edge between the nodes i and j is locally updated once the query is forwarded to node j:

$$\tau = (1 - \varphi) \cdot \tau + \varphi \cdot \tau_0$$

where $\varphi \in (0, 1]$ is the pheromone decay coefficient, τ_0 is the initial value of the pheromone.

Clearly, the update algorithm requires a query message to record the addresses of nodes that have been visited by the message. Due to the scalable locating path, this requirement should not increase the message size dramatically.

As depicted in Figure 8, the update algorithm can remove infrequently updated neighbours from the forwarding table except the succeeding node on the ring topology. In other words, the connection between nodes is dynamically adjusted according to the patterns of object communication or the network conditions. Through this, the topology of the virtual structure evolves at runtime. Furthermore, the size of the forwarding table is not static as contrast to the structured networks. However, it is crucial to maintain the basic ring structure to guarantee the correctness of the locating algorithm.

```
peer.answer(query)
    obtain visited_peers_list from query
    create a reply message
    assign the last peer from visited_peers_list to reply.destination
    send reply to the queried peer
end answer

peer.update(reply)
    for each neighbour in the routing table
        update neighbour.linkPheromone given by Equation 3.6
        if neighbour.linkPheromone < pheromone.MIN
        AND neighbour is not successor
            remove neighbour from the table
        end-if
    end-for

    if reply.precedingNode ≠ NULL
        set reply.destination = reply.precedingNode
        send reply
    end-if
end update
```

Figure 8: An algorithm that updates pheromones via backward propagation. A reply message traverses in the reverse of the querying path.

3.3.4. Proof of correctness

Theorem 2. The locating protocol always locates the queried object given that the basic structure of ARMS is maintained and the hash function correctly produces unique identifiers.

Here, the correctness of the locating protocol of ARMS is analysed through the finite state machine (FSM) as illustrated in Figure 9. Assume the function *query(K,S)* which returns the location information related to an object with name *K* and the resolution state *S*.

First, the source follows one of three transitions in response to the resolution request:

$$query(K,S_1)=\begin{cases} S_3;\ \text{case } a_1 \text{ - predecessor of key K} \\ S_2;\ \text{case } a_2 \text{ - send query via fingers} \\ S_4;\ \text{case } a_3 \text{ - rebuild ARMS fingers} \end{cases}$$

If the identifier of the requested object is in the range of identifiers stored in the succeeding node, the current node becomes the predecessor of the identifier (case a_1). Therefore, the requested object is subsequently found. If the queried identifier resides in the range of a finger (case a_2), the query is forwarded to the subsequent node that has an identifier closer to the request identifier than the current one because of the uniformly directional structure of ARMS. If no condition is met, it implies an incomplete topology and thus a backup function *build_fingers* [23] is activated to re-establish the broken segment of the ring structure (case a_3). The implementation of the function is similar to the algorithm described in [23].

At the intermediate node S_2, the query is forwarded to the next node if the identifier is found in the forwarding table (case b_1). The node becomes the predecessor when the succeeding node contains the target object (case b_2). Otherwise, the backup function is initiated to find the successor node (case b_3).

$$query(K,S_2)=\begin{cases} S_2;\ \text{case } b_1 \ -\ \text{send query via fingers} \\ S_3;\ \text{case } b_2 \ -\ \text{predecessor of key K} \\ S_4;\ \text{case } b_3 \ -\ \text{rebuild ARMS fingers} \end{cases}$$

When the *build_fingers* function is invoked, an update message is issued at the node in the discontinued segment. Once the connection is restored, the query message is sent to the successor of the broken node (case c_1). Thus the search is restored.

$$query(K,S_4)=S_2;\ \text{case } c_1 \ -\ \text{send query via fingers}$$

When a query reaches at the predecessor S_3 of the requested identifier, the target object is thereby found (case d_1) by forwarding the query to its successor node.

$$query(K,S_3)=S_5;\ \text{case } d_1 \ -\ \text{target found}$$

Consequently, the correctness of ARMS locating algorithm is demonstrated as long as the basic ring structure can be maintained. In other words, each node needs to inform only its predecessor node and successor node when it leaves the network.

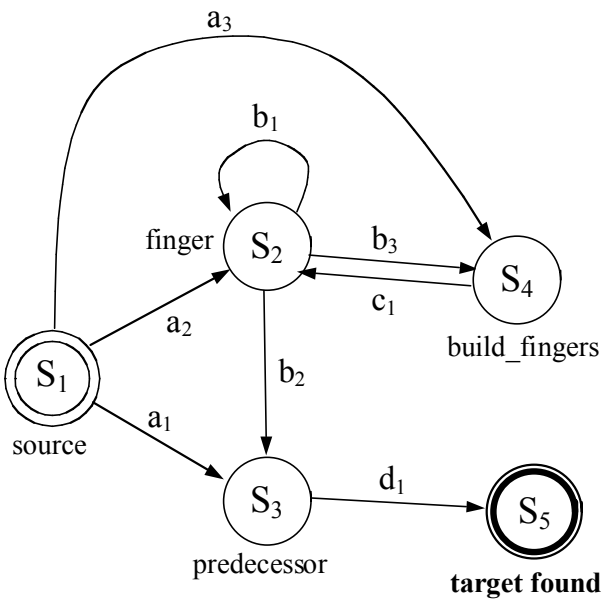

Figure 9: The state diagram of the object locating protocol.

3.4. Performance Improvement via Constructive Heuristic Information

The importance of the constructive heuristic information is to provide guidance for name queries in searching the target and hence a selection of heuristic information is crucial for locating efficiency. The flexibility of ARMS allows various types of heuristic information to be used. For instance, a heuristic that aims at optimising the logical locating length can stimulate the function of structure networks. In other words, a query message is routed to the neighbouring node that has the node identifier closer to the requested object identifier than others.

However, the independency of the virtual structure to the physical network results in a proximity problem, where two nodes are neighbouring on the virtual space may be located far apart from each other in the physical network. Previous research [41, 43, 102] has proposed potential solutions to overcome the proximity problem in P2P networks.

The elegance of ARMS can solve the proximity problem by adapting a new heuristic rule. The heuristic rule aims at routing the query messages with the minimum travelling distance in the physical network. Therefore, a query message is likely routed to the neighbouring node that is physically close-by. The consequence of using such heuristic, however, may be a longer locating path incurred in the object search. Hence, there is a trade off between the physical distance and the logical path on the virtual structure.

Other forms of heuristic information have also been considered for ARMS, such as optimisation for workloads and object locality. In particular, it is important to retain load balancing across the system in order to improve resource utilisation and communication throughputs in face of dynamic communication patterns. Details of the study on the heuristic information are presented at Chapter 6.

3.5. Transparent Object Relocation Scheme

Object migration is the act of transferring a computational or data object between two machines. It is vital for resource management as it enables dynamic load distribution, fault resilience, eased system administration, and data access locality [45]. Hence, migration is a key function in large-scale distributed systems such as Grid or Cloud computing [49, 67, 154-156]. Despite other core functions for achieving object migration, this study has only focused on how to provide transparent accesses to migrated objects here. That is, after migration, the migrated object should be accessible by the same name and mechanisms as if migration never occurred. Such a process is termed *transparent object relocation* [49]. Transparent object relocation is vital for the guarantee of communication continuation.

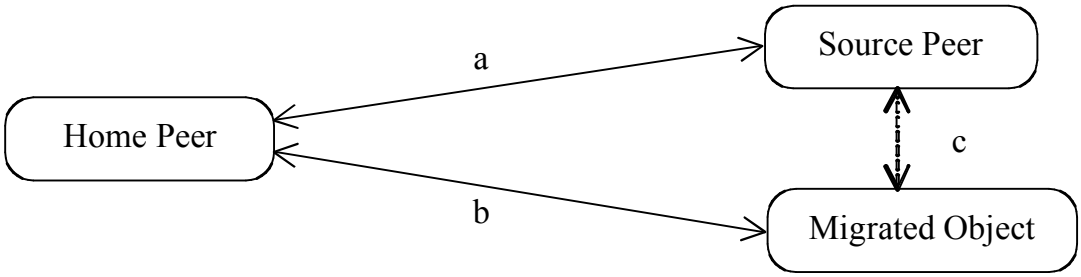

Figure 10: The trombone problem where the distance of links *a*, *b* is much greater than link *c*.

Transparent object relocation can be implemented via one of the four common approaches: *home based*, *forward pointers*, *backtracking* and *broadcast* [34, 49, 138, 139]. With home based, each object is assigned to a fixed server, which is usually the birthplace of that object. Further, the name of the object contains the address of the birthplace and hence messages are transported to the object via the home server. On every migration, an object needs to synchronise with its birthplace in order to update the new location of that object. While the home-based mechanism can simplify the location updating procedure, it may cause large communication overheads unnecessarily incurred if the migration object and the source object of the message are both separated by large distances from the home location, but close to each other. This is commonly known as the *trombone* problem [140] as illustrated in Figure 10.

With forwarding pointers, a migrated object leaves a forwarding address at the last node after migration. Messages subsequently will follow the chain of forwarding pointers to reach the migrated object at the new location. There is only minimum synchronisation required for migration and hence communication can be resumed quicker than the home-based approach does. Because the chain of pointers is growing with the number of object migration, it is not practical for a large number of migrations as it can lead to a long forwarding chain.

Consequently, Emerald [141] uses timestamps to specify the age of forwarding addresses. Through timestamps, the system can replace any outdated forwarding address based on an acknowledgement message that is returned from the final destination. Such a process is also known as backtracking. The backtracking process aims at reducing the overall relocation path by caching the new address of the migrated object at many nodes.

Alternatively, object relocation can be done via either of two broadcast approaches: one which a peer broadcasts query messages to all peers in a network asking for the current location of the migrated object, and one which *change of location* notifications are broadcasted to all peers in a network when migration is completed. Obviously, the broadcast approach does not scale well with a large number of peers and thus it is suitable only for small-scale systems.

In ARMS, every object identifier is assigned to a single peer whose node identifier is nearby on a logical identifier space. It is hence natural to implement the home-based approach for relocating migrated objects in ARMS. However, the approach can cause significant delays in communication due to the synchronisation between the migrated object and the birthplace. To decrease delays incurred in the synchronisation, the ARMS peers keep forwarding pointers for temporarily redirecting messages to the new location during synchronisation with the birthplace. As a forwarding chain can grow in length after each migration, the chain needs to be updated to improve delays incurred in message forwarding. Consequently, a revised update protocol is presented in Figure 11.

In addition, the requesting peer will create new fingers that point to peers along the forward chain. This can improve the likelihood of encountering a forwarding pointer

before the home peer for query messages sent from other peers. This process is also vital for the reliability and the robustness reasons. Again, the pheromone values of those new links will eventually decay to zero when the links are no longer needed.

```
peer.update_search_path(reply)
    if this peer lies on the forward chain
        update forwarding pointer to reply.sourceLocation
    else
        update(reply)
        if peer has not been created in the routing table
          create a temporal finger that points to the peer
        end-if
    end-if
end update_search_path
```

Figure 11: The revised update protocol. It updates both the pheromones of a search path and the forward chain.

```
peer.relocate(key)
    if a forward table contains the key
        /* send the query via the forward pointer */
        set query.destination to be forwardPointer.address
    else
        /* send the query through fingers */
        let C = locate(key)
        send_queries(C)
    end-if
end relocate
```

Figure 12: The object relocation protocol used in ARMS.

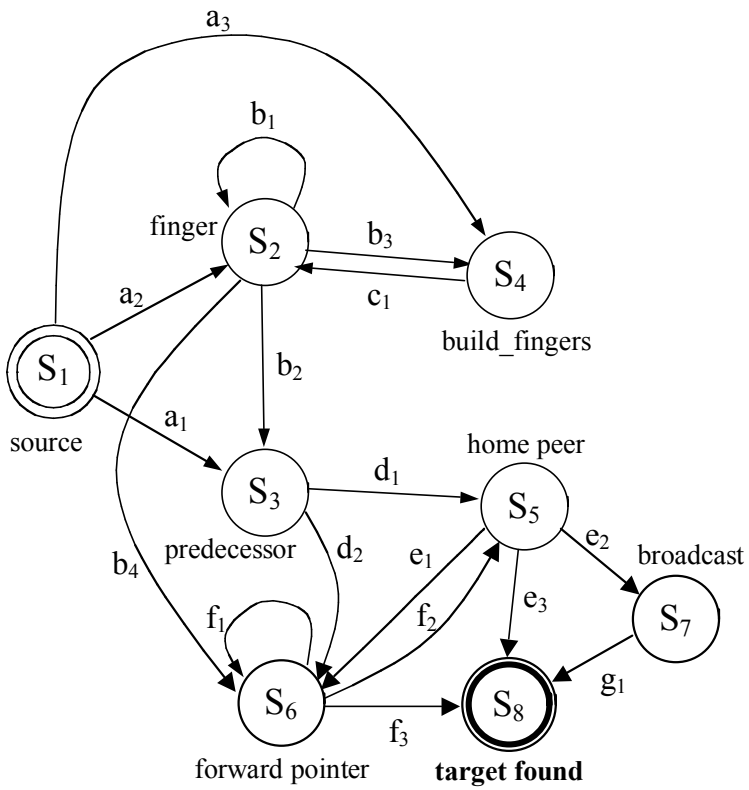

Figure 13: The state diagram of ARMS relocation scheme.

Figure 12 illustrates the ARMS relocation algorithm, which it is extended from the locating scheme shown in Figure 7. By switching from fingers to forward pointers, ARMS can keep track of migrated objects in the system. Two new states are subsequently added to the locating protocol as illustrated in Figure 13.

When a query arrives at a peer lying on the forward chain, the query is subsequently sent through the forward chain as illustrated in case b_4, case d_2 and case e_1.

$$query(K,S_2)=\begin{cases} S_2;\ \text{case } b_1 \ -\ \text{send query via fingers} \\ S_3;\ \text{case } b_2 \ -\ \text{predecessor of key K} \\ S_4;\ \text{case } b_3 \ -\ \text{rebuild ARMS fingers} \\ S_6;\ \text{case } b_4 \ -\ \text{send query via forward pointers} \end{cases}$$

$$query(K,S_3) = \begin{cases} S_5; \text{ case } d_1 \text{ - locate home peer of key K} \\ S_6; \text{ case } d_2 \text{ - send query via forward pointers} \end{cases}$$

The current location of the requested object can be found through either the home peer of the object (case e_1) or the forward chain (case f_3). Generally, the home peer is synchronised with the migrated object so that it keeps the current location of the object. During the synchronisation, however, the forward chain is used temporarily to redirect all messages that target the migrated object to the new location. The length of the forward chain is proportional to the number of migrations and thus a frequently migrating object may leave a long forward chain behind. Because the length has impact on the efficiency of the relocation protocol, the update mechanism is essential to reduce the long chain. As a result, the expected relocation path is on the order of log(N) in most cases. Nonetheless, frequent migration should be avoided as it incurs significant migration overheads, which defeats the purpose of the object migration.

$$query(K,S_5) = \begin{cases} S_6; \text{ case } e_1 \text{ - send query via forward pointers} \\ S_7; \text{ case } e_2 \text{ - broadcast queries to all peers} \\ S_8; \text{ case } e_3 \text{ - target found} \end{cases}$$

$$query(K,S_6) = \begin{cases} S_6; \text{ case } f_1 \text{ - send query via forward pointers} \\ S_5; \text{ case } f_2 \text{ - send query to the home peer} \\ S_8; \text{ case } f_3 \text{ - target found} \end{cases}$$

$$query(K,S_7) = S_8; \text{ case } g_1 \text{ - target found}$$

When a peer along the forward chain is departed, i.e. the chain is broken (case f_2), a query will be returned to the home peer of the requested object. If the home peer knows the current address of that object, the query will be redirected to the destination.

Otherwise, if the address is unknown to the home peer, a broadcast mechanism is needed to locate the target. Due to the synchronisation between the home peer and the migrated object, the latter case should rarely happen.

3.6. A Comparison of ARMS and Other Peer-based Naming Systems

While the design of ARMS is largely inspired by naming systems based on randomised overlay, it is unique in many ways, especially in its direct supports for message passing between objects. In most randomised overlay systems, their object locating schemes [35, 41, 43, 104, 106] are still primitive as they are mainly based on greedy routing, where a decision is made based on the locally optimal strategy at every step. Obviously, one of the key problems of the greedy algorithms is that they mostly fail to find a global optimal solution although they offer faster computation than other techniques. More importantly, the greedy routing is stateless and lacks adaptability. The greedy routing does not memorise any past solutions and thus it cannot utilise the solutions that are known to be useful. This restricts the support for dynamic patterns of object communication. Furthermore, object migration has not been considered by those naming systems; it is explicitly supported in ARMS.

Contrary to most of the structured and randomised naming systems, ARMS always stores the object identifier on the node that creates the object. Any message that embeds with the object identifier can be routed directly to the owner node, without the need for suspending the message passing during the name resolution. As a result, the overall delays incurred in the message passing in ARMS is $D_{resolution}$, instead of $D_{resolution} + D_{reply} + D_{routing}$ – as characterised by most naming systems.

Randomised-Chord [104] and ARMS are both developed based on the Chord naming system. However they are differed in the designs of the routing tables. ARMS offers

more flexibility as it allows more than one link (finger) to be included at every range. The use of redundant fingers aims to reducing the network diameter and to improving the latency resiliency.

ARMS and AntCAN [105] share many similarities, including the use of a structured network as a substrate for searching, link exploitation via an ACO algorithm, and feedbacks using reply messages. Nevertheless, the two models are substantially different in the following aspects.

- *Flexibility of Chord vs. rigidness of CAN*. AntCAN is based on CAN, whose neighbours are determined by the coordinate space with fixed degrees. Hence it is not suitable to adapt a proximate routing scheme [120]. Gumnadi *et al.* [43], on the other hand, observed that the Ring geometry has unsurpassed flexibility compared to other geometry and provides natural support for proximity metrics. In short, the AntCAN places a burden on the effectiveness of ACO algorithms while ARMS takes advantage of the flexible Chord ring as restructuring the overlay network through ACO with respect to dynamic communication patterns.

- *ARMS preserves the efficiency of the Chord*. Because the ARMS routing table is organised in a predictable manner like the Chord, its expected routing path is O(log$N$). However, it is not clear the routing performance of the AntCAN since the cache that keeps additional contacts is not specifically organised. In general, the search length of the CAN grows much faster with the number of nodes than other structured models.

- *Emphasis of exploitation and exploration*. In ARMS, various forms of heuristic information have been examined to tune the performance of object location, such as proximity, load balancing and data correlation. The heuristic information works in parallel with the pheromone values that exploit past solutions to improve the routing performance and the adaptability.

- *Explicitly support for object migration*. ARMS explicitly provides support for transparently locating migrated objects. However, object migration has not been considered in the AntCAN.

In summary, ACO is primarily used to refresh the contacts of the AntCAN, which is similar to a cache management. However, ARMS applies ACO in four equally important functions: *path exploitation, network exploration, overlay reconstruction* and *migration support*.

Similar to the DOLR API [120], ARMS can provide scalable location independent routing. However, the DOLR API is designed for an Internet-scale, highly available storage utility and hence the resiliency to node arriving and departing is a centre of focus in such a model. On the other hand, ARMS aims at reducing latencies incurred in locating objects and it is achieved via an adaptive location algorithm, which is able to exploit the existing paths or to explore alternative paths in response to any change in the network conditions. In addition, ARMS allows every node to flexibly expand its routing table to reduce the diameter of the overlay network, which leads to a shorter lookup path. At the same time, it can keep the average table size small by automatically removing unused or inefficient entries from the routing table.

4. Analysis of Patterns in Object Communication

4.1. Selection of Java-based Benchmark Programs

The adaptive locating protocol of ARMS is designed to facilitate the patterns of communication within object-oriented software. This chapter focuses on the study of those patterns. A set of object-oriented benchmark programs was chosen – *AutoFocus*, *DynamicJava*, *ImageJ* and *Rhino*. These are all implemented in Java, one of the more important and popular object oriented programming languages [3]. Results obtained in the analysis can be considered as typical patterns given Java is a fundamental form of object oriented language.

The four chosen benchmark programs are summaries below.

- *AutoFocus* [142] is a Java-based desktop search engine used to locate information on the PC, network disks, mail boxes, websites and AutoFocus Server sources. The notable features of the AutoFocus include the use of facets that allow users to find documents on more than just keywords, and the use of Cluster Maps that allows users to visualise the relationship among files, web pages or emails.

- *DynamicJava* [143] is a Java source interpreter. It executes Java programs specified by the Java Language Specification plus scripting features. The applications of the DynamicJava include an extension language that allows the user to configure or to extend applications, a debugging tool that helps the user to test an application in an interactive way, and a rapid prototyping tool.

- *ImageJ* [10] is a Java-implementation image processing and analysis program, featuring portability, multithreading and open source architecture. It can perform processing of many common image formats including TIFF, GIF, JPEG, BMP, DICOM, FITS and "raw". Furthermore, it can display temporal or spatial related images in a single window, where the set of images is called a stack. ImageJ offers a wide range of image processing and analysis tools such as smoothening and sharpening, adding and removing noise, distance measurement and displaying density histograms.

- *Rhino* [7] is an open-source implementation of JavaScript engine written entirely in Java. It converts JavaScript codes into Java classes using two approaches: namely, compiled mode and interpreted mode. Due to its popularity, the Rhino engine has been bundled in Java Standard Edition (SE) version 6 [144], which was released by Sun Microsystems. This makes it easier to integrate JavaScript as part of Java programs, and to access Java resources from JavaScript.

Next, the characteristics of object communication were studied via a profiler. Generally, profilers [145] are commonly used to obtain information of runtime execution of a program. Most profilers [145, 146] can provide comprehensive information on CPU performance profiling, method invocation profiling, memory profiling and thread profiling. In this study, the developed profiler is based on a dynamic instrumenting model [145, 147]. The profiler provided vital information for studying the patterns of Java objects and their communication based on four benchmark programs. Section 4.2 examines the results of the profiling study.

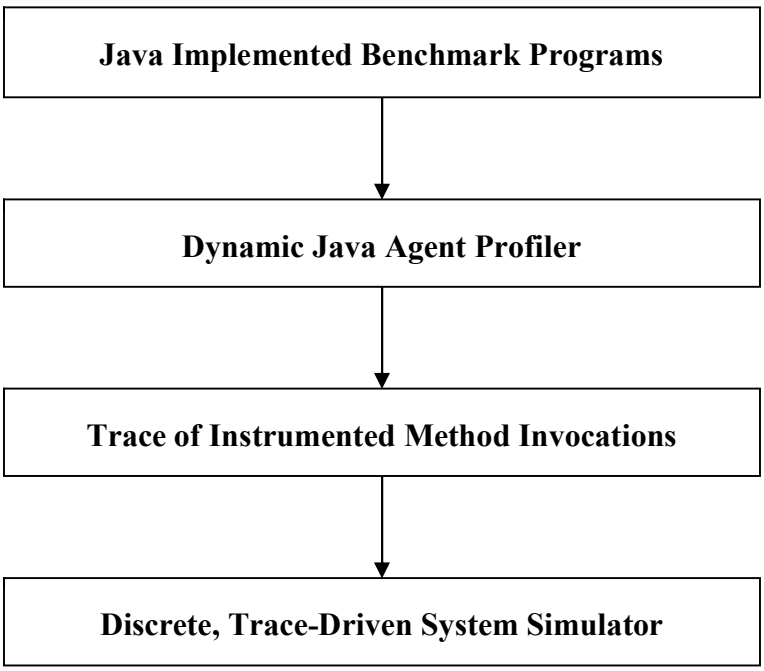

Figure 14: The flow of the methodology adopted in this study.

The profiler can produce traces of instrumented method invocations from four Java-based benchmark programs. These instrumented method invocations represent message passing between objects. Due to the aims of this study, local computations such as arithmetic operations, I/O calls and conditional executions were ignored by the profiler. In the third stage of this study, traces attained from the benchmark programs were supplied to a discrete-event, trace-driven simulator for evaluating the performance of the proposed naming model. The architectural design of the simulation is described in Appendix. Figure 14 above summaries the methodology of the study.

4.2. Characteristics of Objects and Their Message Passing

4.2.1. Object dynamics

In most object-oriented programming languages, objects are constantly created and removed at runtime. Figure 15 illustrates the distribution of the size of object populations based on the benchmark programs. It is noted that the x-axis of the figure is a normalized execution cycle and the y-axis is the ratio of sampling population to the

number of totally created objects during execution. Clearly, the size of the sampling population was relatively low on average. The average population sizes were 10% – Rhino, 16% – ImageJ, 8% – AutoFocus and 5% – DynamicJava. The small values were due to a great deal of objects that had the short lifespan. As illustrated in Table 3, about 69% of the overall of the AutoFocus objects lived as long as at most 5% of the total execution time. The portions of the shortly lived objects were also high for other benchmarks. The similar results have also been observed in [148].

Furthermore, the large amount of shortly lived objects caused a strong correlation between the object creations and *potential* removals as demonstrated from Figure 16 to Figure 19. The potential removal refers to the moment whenever an object could be potentially garbage collected as it was no longer in use by the program. The figures suggest that there is a need for naming support for concurrent binding and unbinding to run these programs efficiently.

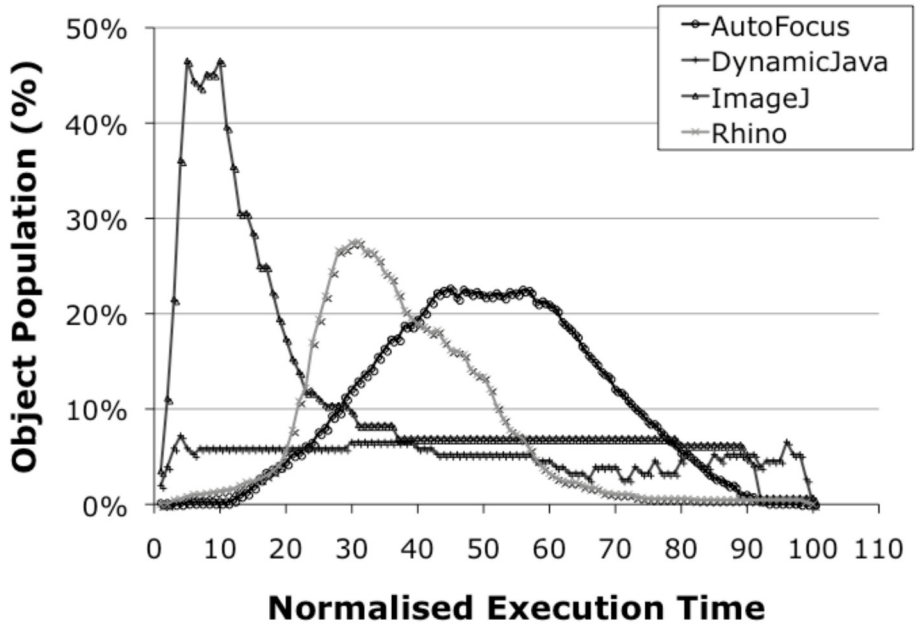

Figure 15: A demonstration of the percentage of the sampling object population to the overall population for four benchmark programs as the progress of the execution.

Normalised Execution Time	*Java Benchmark Programs*			
	AutoFocus	**DynamicJava**	**ImageJ**	**Rhino**
5%	69.0%	87.6%	58.3%	59.2%
10%	4.1%	2.6%	13.9%	13.7%
15%	3.9%	0.7%	8.3%	9.2%
20%	1.8%	0.0%	6.9%	6.9%
25%	2.9%	0.0%	2.1%	3.7%
30%	2.8%	2.6%	2.1%	2.1%
35%	3.4%	1.3%	0.0%	1.7%
40%	3.3%	2.0%	1.4%	1.4%
45%	1.8%	0.0%	0.0%	0.9%
50%	1.7%	0.0%	0.0%	0.2%
55%	0.9%	0.0%	0.0%	0.1%
60%	1.8%	1.3%	0.0%	0.2%
65%	1.5%	0.7%	0.0%	0.1%
70%	0.9%	0.0%	0.0%	0.0%
75%	0.2%	0.0%	0.0%	0.0%
80%	0.0%	0.0%	0.7%	0.1%
85%	0.0%	0.0%	0.0%	0.1%
90%	0.0%	0.0%	4.9%	0.0%
95%	0.0%	1.3%	0.7%	0.0%
100%	0.0%	0.0%	0.7%	0.2%

Table 3: An illustration of the distribution of the lifespan of Java objects. The life span is measured as the percentage of the total execution time.

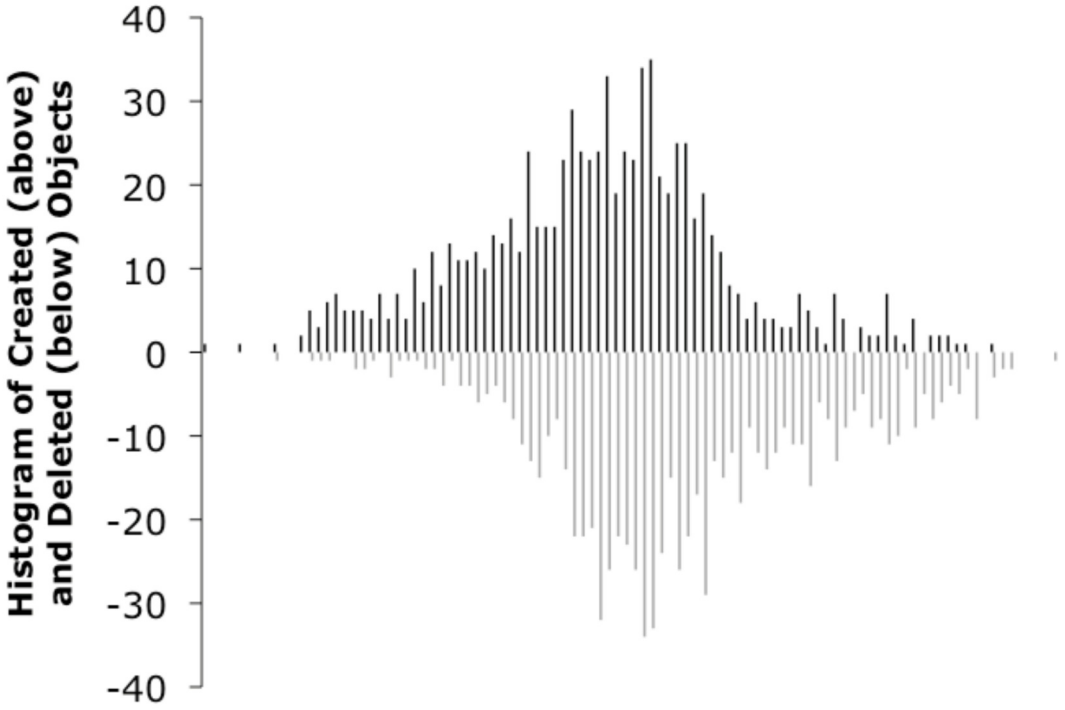

Figure 16: An illustration of the number of object creations [above] and potential object deletions [below] as the function of the normalised execution time in the Autofocus.

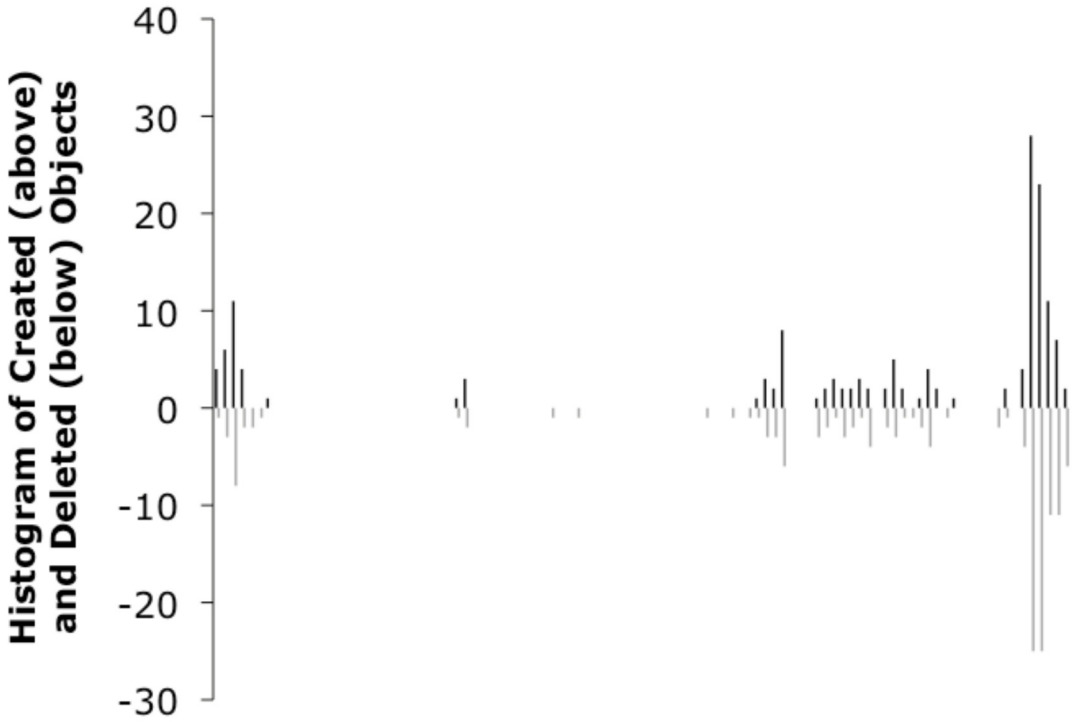

Figure 17: An illustration of the number of object creations [above] and potential object deletions [below] as the function of the normalised execution time in the DynamicJava.

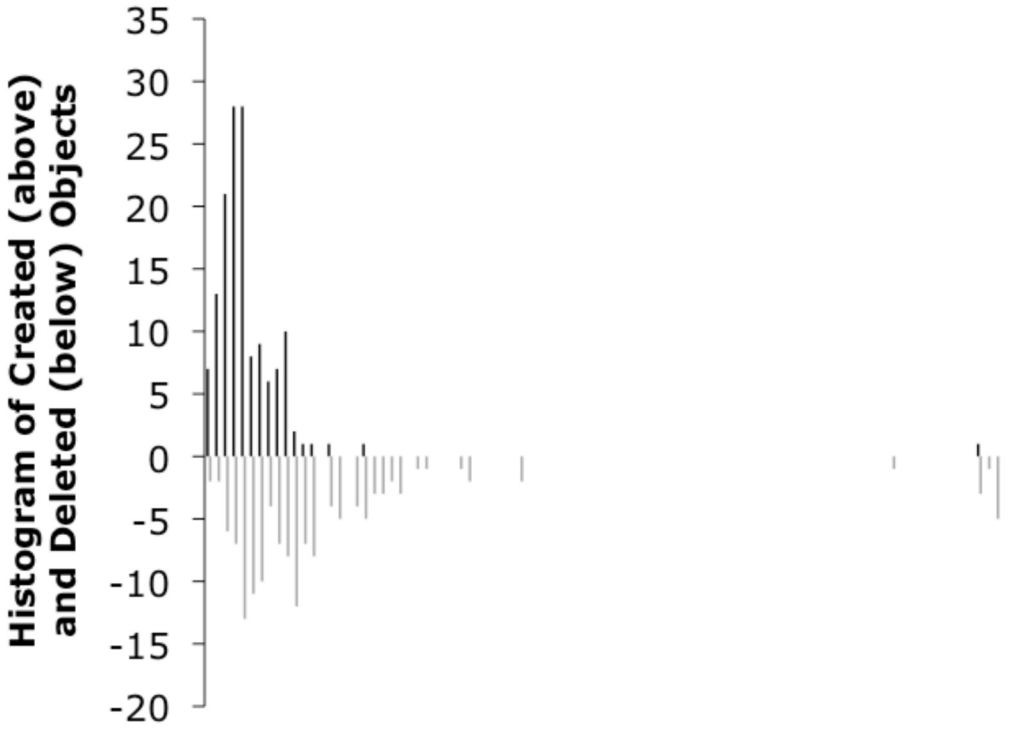

Figure 18: An illustration of the number of object creations [above] and potential object deletions [below] as the function of the normalised execution time in the ImageJ.

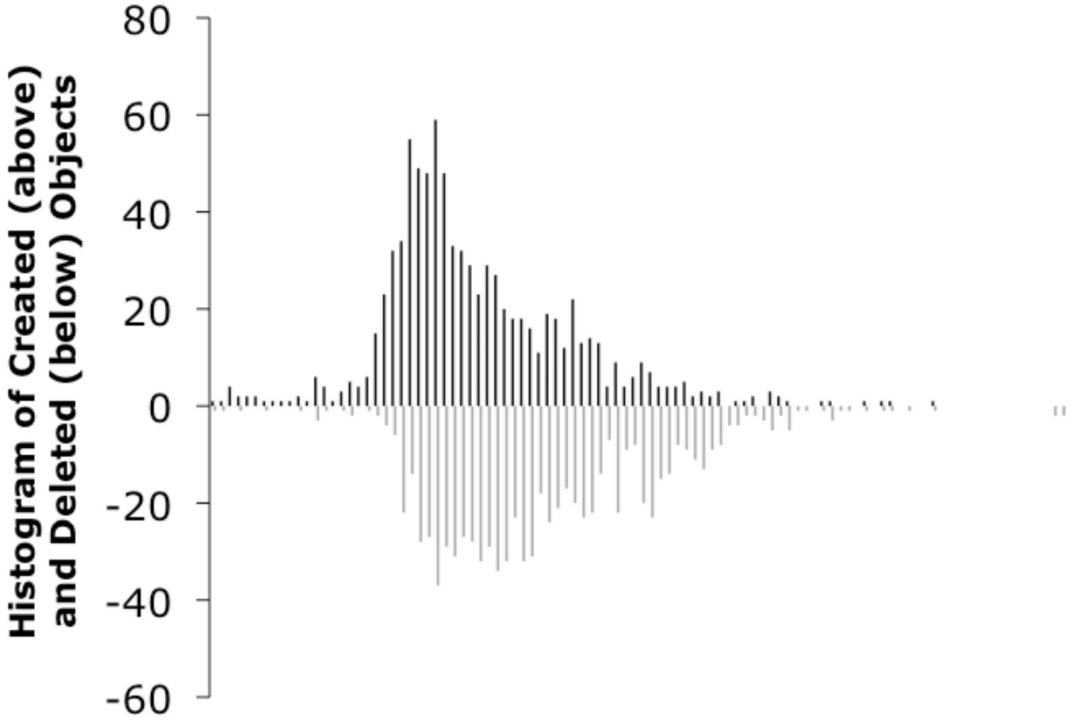

Figure 19: An illustration of the number of object creations [above] and potential object deletions [below] as the function of the normalised execution time in the Rhino.

4.2.2. Non-uniform object communication

The profiling study observed that the message workloads were not evenly distributed amongst objects. That is, a few objects sent the large amount of messages whereas most objects sent just a few messages. As shown in Table 4, the number of invocations sent by the majority of objects was less than 10% of the total invocations. Furthermore, there was a great deal of objects that did not send out any invocation. These objects merely performed local operations such as writing data to a local file. Furthermore, this study observed that the lifespan of an object was correlated with its message workloads. That is, the objects received or sent more messages resided in the system for a longer period.

The objects can be categorised into two classes according to the number of messages sent or received. Hot objects are those sent or received more than 50% of the overall messages while the others are termed cold objects. Generally, hot objects tend to establish more connections to other objects than cold objects. The profiling results showed that there was a positive correlation between the number of sent invocations and the number of received invocations for an object. In other words, objects that sent more invocations are likely to receive more invocations.

4.2.3. Exploring locality in object communication

The control structures of a program can affect the patterns of the object communication. These structures include iterative construction and switch construction. To illustrate this, the source code of a class, termed **IJ**, within the ImageJ benchmark program is listed in Figure 20. The *setPixels()* method of the **ip** object will be invoked *N* times through the for-loop. Even if the loop is only executed once, the **ip** object will still be accessed several times by the **IJ** class.

Total Number of Invocations	*Benchmark Programs*							
	AutoFocus		**DynamicJava**		**ImageJ**		**Rhino**	
	Sent	*Received*	*Sent*	*Received*	*Sent*	*Received*	*Sent*	*Received*
5%	58.2%	0.0%	49.7%	0.0%	19.4%	0.0%	47.8%	0.0%
10%	36.9%	95.7%	47.1%	97.4%	78.5%	97.2%	51.7%	99.9%
15%	2.2%	2.6%	2.6%	2.0%	1.4%	2.1%	0.5%	0.1%
20%	1.0%	0.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
25%	0.5%	0.6%	0.0%	0.7%	0.0%	0.0%	0.0%	0.0%
30%	0.4%	0.0%	0.7%	0.0%	0.0%	0.0%	0.0%	0.0%
35%	0.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
40%	0.1%	0.0%	0.0%	0.0%	0.0%	0.7%	0.0%	0.0%
45%	0.1%	0.1%	0.0%	0.0%	0.7%	0.0%	0.0%	0.0%
50%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
55%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
60%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
65%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
70%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
75%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
80%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
85%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
90%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
95%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
100%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Correlation	***0.296***		***0.637***		***0.102***		***0.545***	

Table 4: A demonstration of the frequency of invocations sent or received by objects in each benchmark program. The last row shows the correlation of the number of sent messages and the number of received messages by the same object in a benchmark.

	AutoFocus		**DynamicJava**		**ImageJ**		**Rhino**	
Parameter	*Amount*	*Percent*	*Amount*	*Percent*	*Amount*	*Percent*	*Amount*	*Percent*
Total object calls	2623	100%	531	100%	4636	100%	4417	100%
Class calls	0	0.0%	9	1.69%	1751	37.77%	21	0.48%
Inheritance calls	84	3.20%	228	42.94%	336	7.25%	765	17.32%
Temporal calls	519	19.8%	101	19.0%	2866	61.8%	837	18.9%

Table 5: An illustration of the profiling results for the total number of object calls, the number of the class calls, the number of inheritance calls, the number of temporal calls.

```
for ( int i =1; i <= n; i++ )
{
        ip.setPixels ( stack.getPixels( i ) );
        ip.setMask ( mask );
        ip.setRoi ( r );
        ip.setCalibrationTable ( cTable );
        if ( doMasking )
                ip.snapshot ();
        ( ( PlugInFilter ) theFilter ).run ( ip );
        ...
}
...
```

Figure 20: An example of source code of the **IJ** class contained in the benchmark ImageJ.

Such a property is well known as *locality of reference* in the discipline of computer science. There are two common types of locality of reference: temporal and spatial. Temporal locality refers to a resource that is referenced at one point in time will be referenced again sometime in the near future. Spatial locality, on the other hand, is the concept that the likelihood of referencing a resource is higher if a resource near it has been referenced.

Table 5 depicts the statistics of two types of locality observed in four benchmark programs. Firstly, there are two types of spatial invocations: cluster invocation and inheritance invocation. Firstly, the cluster invocations refer to calls between the objects of the same type. That is, these object are instantialised from the same class. For instance, the number of cluster invocations found in ImageJ benchmark was 37.77%.

Secondly, the inheritance calls refer to invocations between an object and its inheriting objects. Inheritance is one of the fundamental concepts in OOP models. It is a powerful

mechanism to extend the functionality of an object naturally. For the DynamicJava, four out of ten invocations between objects were inheritance invocations. Profiling results also revealed a moderate amount of inheritance invocations in the Rhino benchmark.

Three of the four benchmarks demonstrated the characteristics of spatial locality in their object communication. The AutoFocus, in contrast, did not show any significant number of spatial calls. This implies that object communication was rather scattered in the benchmark. On the other hand, the AutoFocus, along with other benchmarks, revealed the notable amount of temporal invocations.

4.3. Discussion on Profiling Results

Firstly, objects are not allocated in a static manner in object-oriented software. Indeed, they are dynamically created in the system via the creation message. Because messages cannot be processed by an object until the creation procedure of that object is completed, the performance of the creation procedure has impact on the efficiency of message passing. Furthermore, objects were created regularly and most of them could be garbage collected shortly after the creation. That is, most objects had short life spans in execution. When an object is garbage collected, an empty spot will be created in the system. It is desired to fill up the empty holes with newly created objects or existing objects to reduce the average distance between objects and to improve resource utilisation.

The message workloads of objects were uneven during the execution. The majority of objects sent (received) only a few messages whereas a few objects sent (received) most messages. Further, objects that sent more invocations are likely to receive more invocations. The unbalanced workloads may cause query hotspots and routing hotspots in the network. It is possible to handle query hotspots with caching while a more liberal

routing scheme is required to overcome the problem of routing hotspots. On the other hand, the hot objects could be useful in name resolution. There could be advantageous to begin the search of a requested object with the set of hot objects and then to expand the search to cold objects because;

- Hot objects have a higher chance to be the requested object than cold objects do;

- Hot objects are likely to maintain more object connections; and

- It is more efficient to search a hot object than a cold object due to the small population of those hot objects in the system.

Thirdly, the locality properties such as spatiality were observed in message passing between objects. Similar to hot objects, these properties are important for optimisation purposes. On performing name resolution in a distributed environment, it is fundamental to locate the node that stores the address of the requested object. The process can be improved by deliberately searching nodes that hold objects related to the requested object.

Finally, dynamic object creation and object addresses in messages imply a variable topology formed by the objects. Such a variable topology can subsequently lead to varied patterns of object communication at runtime. It is desired to have a naming scheme that adapts to the dynamic patterns to achieve the better performance in message passing.

5. Simulation Study of the Performance Characteristics of the Adaptive Locating Protocol

5.1. Performance Measures

The locating protocol of ARMS is driven by the distributed Ant Colony Optimisation (ACO) technique. The use of ACO is twofold: to exploit the good solutions that were discovered previously, and to explore alternative solutions when the network conditions have changed. This chapter examines the exploitation feature of the ARMS locating protocol and the dynamic path exploration in ARMS is examined in the next chapter. In particular, this chapter presents a study of the characteristics of the ACO parameters and their impacts on the ARMS locating efficiency.

This study was performed through a simulator described in Appendix. The simulation environment used in this chapter was based on a two-dimensional grid structure since the similar structure is commonly used in cluster computing systems. For the sake of simplicity, the latency of the local computation was modelled by a normal distribution function, as the latency is trivial compared to the communication latency.

The inputs of the simulation were traces of invocations exchanged between objects, which had been extracted from four Java benchmark programs: AutoFocus [142], DynamicJava [143], ImageJ [10] and Rhino [7] as discussed in the previous chapter. The simulation measured two key outputs: *expected lookup length* and *simulation completion time*. Firstly, the resolution process is done via forwarding query messages among neighbouring nodes. The process continues until the query message reaches the node holds the current address of the queried object. The distance travelled by the query message is denoted the length of the search path, or the lookup length, or the *length* in

short. The length is commonly used as one of the key performance measurements in the study of distributed lookup and it is hence used here to evaluate the efficiency of the proposed locating scheme. Without the loss of clarity, the values of the lookup length are referred to the *expected values* in the study. However, other studies [41, 43] argued that the lookup length is not necessarily the single factor that affects the system's performance. Factors like proximity and work balance are also vital to the search performance. In spite of these issues, this chapter mainly focuses on the effect of the lookup length on the system's efficiency here while study of different parameters is presented in Chapter 6.

Secondly, the simulation completion time refers to the duration of executing a trace file in the simulator. Since the efficiency of object communication is inversely related to the latency of the naming process and the routing process, the quicker naming and routing process will lead to faster communication between objects. As each communication is faster, the overall simulation time will be shortened. Thus, the simulation completion time can compare how efficiently two naming processes perform as the routing process was fixed in this study. It is noted that units for the completion time are based on a tick produced by the discrete simulator and therefore they do not imply the real performance of applications.

The rest of the chapter is organised as follows. The next section shows the investigation on the parameters that directly or indirectly affect the computation of the probability process used in the ACO algorithm. Next, the scaling property of the proposed scheme was studied and presented in Section 5.3. There are four categories of tests: the size of the forwarding table, the network size, the amount of parallel query messages used in a

search, and the object populations. Finally, a different network topology was used to test the ARMS locating algorithm. Simulation results are shown in Section 5.4.

5.2. Investigation of Parameter Settings

5.2.1. Test description

The parameters that contribute to the computation of the probability in ACO were investigated with respect to the length of the search path and the completion time. The examined parameters included the pheromone importance factor (PIF or α), the heuristic importance factor (HIF or β), the evaporation rate (ER or ρ), the decay rate (DR or φ), the heuristic constant (HC or Q_H), and the pheromone upper bound (PUB or τ_{max}).

Several values for every parameter were tested while the rest are being kept constant on a 2D grid network of the size 40 by 40. The default values of the parameters were $\alpha = 1$, $\beta = 1$, $\rho = 0.3$, $\varphi = 0.3$, $Q_H = 10$, $\tau_{min} = 1$ and $\tau_{max} = 10$. Furthermore, several simulations for each configuration were performed in order to obtain mean values.

The combinations of values that were tested with the simulator are listed as follows.

- $\alpha \in \{2, 2.5, 5, 8, 10, 15, 20, 25, 30\}$
- $\beta \in \{1.5, 2, 2.5, 5, 8, 10, 15, 20, 50\}$
- $\rho \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$
- $\varphi \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$

- $Q_H \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$

- $\tau_{max} \in \{1, 4, 8, 10, 12, 16, 20\}$

Test results have been organised into two tables. First, Table 6 outlines the lookup lengths as the function of each parameter. Secondly, Table 7 summarises the execution time. The results for the simulation of each benchmark program were separately reported because they responded to the parameters differently. Furthermore, the underlined values were the best performance results under their categories.

5.2.2. Pheromone importance factor (PIF)

This factor determines the influence of the pheromones in the probability computation. Choosing the higher α value will increase the exploiting ability of the algorithm while decrease its exploring ability. Intuitively, α parameter encourages utilising existing solutions and thus can lead to small variations in path establishment. This is particularly useful for message passing with significant temporal locality.

The simulation study varied the value of PIF between 2 and 30 and obtained the average length of the object locating path and the simulation completion time. The average length and the simulation time for four benchmarks are presented at the first row in Table 6 and Table 7. Test results have been plotted in Figure 21 and Figure 22.

With benchmarks AutoFocus, DynamicJava and Rhino, the locating path expanded as the increase of α. As shown in figures, the optimal values of α were falling between 2.5 and 8 for those benchmarks.

However, the length of the locating path obtained from the ImageJ benchmark tended to decline along the x-axis, where the optimum α was at 20. Such an exceptional behaviour is largely due to the higher percentage of temporal calls found in ImageJ profiling traces. Those temporal calls could take advantage of the α parameter by traveling with the established paths. Therefore, benchmarks with a smaller portion of temporal calls performed worse with the higher α values.

Figure 22 revealed a growing trend in the execution completion time for all benchmarks as the function of α value. The lowest points occurred between 2 – 2.5. As explained previously, the completion time was directly influenced by the efficiency of the locating algorithm. The conflicts between the plots of locating path and the plots of completion times imply that the lookup length is not necessarily the single factor that affects the locating efficiency. These results have agreed with the observations made by Castro and Gummadi [41, 43], where the physical distance of the querying path is sometimes more important than the expected lookup step. However, most P2P models have just emphasised on the expected length for the performance study of those models. On the other hand, the succeeding chapter will study other factors that likely have impacts on the locating performance, such as proximity and load balancing.

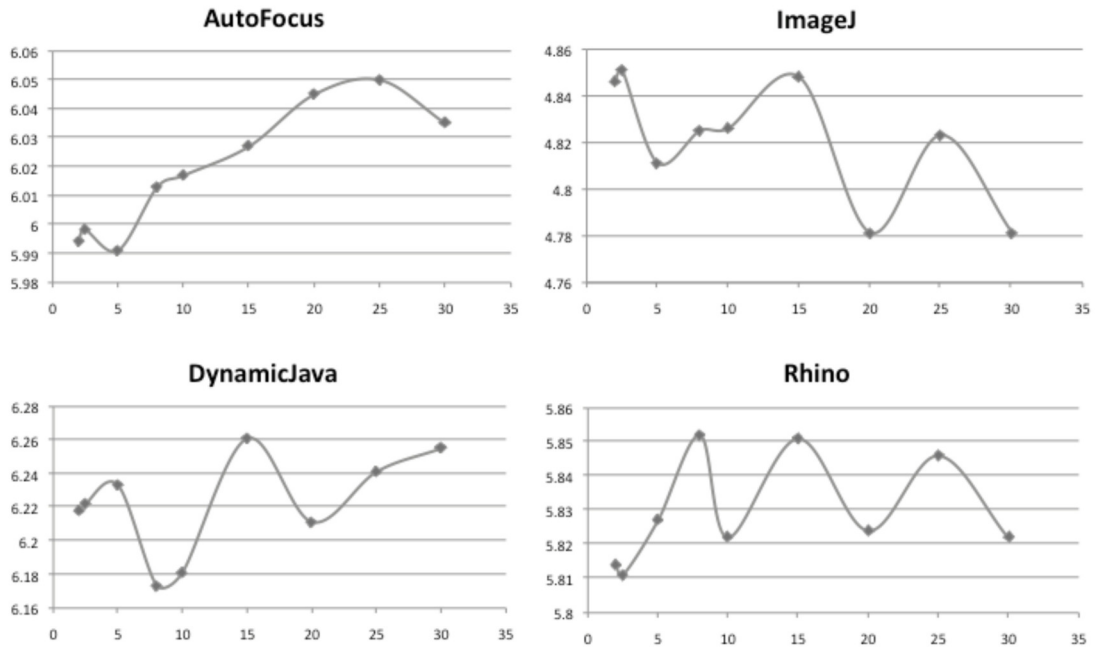

Figure 21: The pheromone important factor versus the search path for four benchmarks.

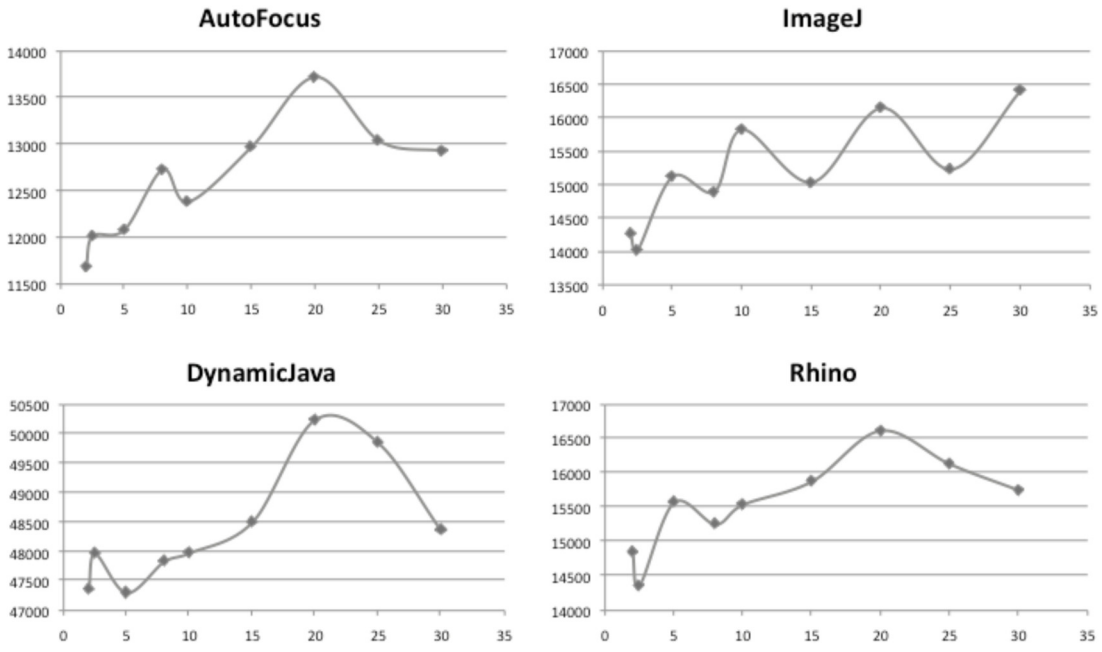

Figure 22: The pheromone importance factor versus the simulation completion time for four benchmarks.

Parameter	Benchmark	Test Values								
		2	**2.5**	**5**	**8**	**10**	**15**	**20**	**25**	**30**
α (PIF)	*AutoFocus*	5.994	5.998	**5.991**	6.013	6.017	6.027	6.045	6.050	6.035
	DynamicJava	6.218	6.222	6.233	**6.173**	6.181	6.261	6.211	6.241	6.255
	ImageJ	4.846	4.851	4.811	4.825	4.826	4.848	**4.781**	4.823	4.781
	Rhino	5.814	**5.811**	5.827	5.852	5.822	5.851	5.824	5.846	5.822

		1.5	**2**	**2.5**	**5**	**8**	**10**	**15**	**20**	**50**
β (HIF)	*AutoFocus*	5.896	5.821	5.761	5.562	5.504	5.473	5.440	5.425	**5.357**
	DynamicJava	6.139	6.036	5.976	5.784	5.679	5.678	5.656	5.647	**5.618**
	ImageJ	4.926	4.847	4.717	4.599	4.600	**4.462**	4.469	4.541	4.524
	Rhino	5.733	5.662	5.617	5.465	5.381	5.365	5.346	5.326	**5.303**

		0.10	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**
ρ (ER)	*AutoFocus*	**5.886**	5.911	5.990	6.006	6.060	6.115	6.131	6.147	6.272
	DynamicJava	**6.199**	6.234	6.246	6.231	6.222	6.293	6.274	6.273	6.366
	ImageJ	4.888	4.996	4.832	4.852	**4.747**	4.828	4.827	4.970	4.886
	Rhino	**5.720**	5.747	5.810	5.877	5.936	6.008	6.017	6.031	6.118

		0.10	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**
φ (DR)	*AutoFocus*	5.982	5.980	5.990	5.977	5.987	5.971	5.969	**5.968**	5.989
	DynamicJava	6.225	6.215	6.246	**6.190**	6.242	6.235	6.233	6.212	6.244
	ImageJ	4.909	5.000	**4.832**	4.931	4.874	4.973	4.949	4.893	4.934
	Rhino	5.813	5.811	5.810	5.815	5.810	5.806	5.824	**5.803**	5.815

		1	**4**	**8**	**10**	**20**	**50**
Q_H (HC)	*AutoFocus*	**5.964**	5.965	5.978	5.990	5.964	5.981
	DynamicJava	6.241	**6.205**	6.257	6.246	6.263	6.233
	ImageJ	4.860	4.940	4.956	**4.832**	4.936	5.023
	Rhino	5.820	5.810	5.823	**5.810**	5.822	5.823

		5	**10**	**15**	**20**	**50**	**100**
τ_{max} (PUB)	*AutoFocus*	6.004	5.990	5.964	5.945	5.910	**5.899**
	DynamicJava	**6.199**	6.246	6.262	6.211	6.217	6.258
	ImageJ	4.834	**4.832**	4.946	4.973	4.970	4.951
	Rhino	5.871	5.810	5.790	5.757	5.718	**5.709**

ABBREVIATIONS

PIF – Pheromone Importance Factor controls the weight of pheromone.
HIF – Heuristic Importance Factor controls the weight of heuristic information.
ER – Evaporation rate determines the evaporating rate of pheromone at each update.
DR – Decay rate determines the decaying rate of pheromone after every computation.
HC – Heuristic Constant is used to update the concentration of pheromone.
PUB – Pheromone Upper Bound controls the upper limit of pheromone.

Table 6: Impact of the different configurations on the length of the lookup path for four benchmark programs. The shortest path length in each category has been highlighted.

Parameter	Benchmark	Test Values								
		2	**2.5**	**5**	**8**	**10**	**15**	**20**	**25**	**30**
α (PIF)	*AutoFocus*	**11686.9**	12014.4	12083.3	12728.5	12383.4	12977.3	13724.3	13037.7	12930.0
	DynamicJava	47352.2	47965.0	**47300.2**	47832.3	47976.7	48502.7	50245.0	49848.2	48358.8
	ImageJ	14272.1	**14021.8**	15120.5	14896.2	15826.5	15037.1	16150.3	15230.8	16419.3
	Rhino	14854.1	**14365.7**	15574.2	15258.3	15542.4	15880.7	16620.5	16134.8	15752.8

		1.5	**2**	**2.5**	**5**	**8**	**10**	**15**	**20**	**50**
β (HIF)	*AutoFocus*	12089.2	11573.9	11672.1	11529.5	**11207.7**	11876.3	11706.3	11800.1	12017.2
	DynamicJava	47369.3	45237.9	46084.8	45655.0	45547.5	45713.0	**45022.6**	46306.3	48234.8
	ImageJ	**12797.2**	13355.4	14092.4	14928.2	12911.3	13517.9	13885.4	12961.0	13200.0
	Rhino	14417.4	14354.4	14384.3	14668.8	14296.2	14878.4	**14181.2**	15129.5	16018.5

		0.10	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**
ρ (ER)	*AutoFocus*	11787.4	12233.9	11932.1	11827.5	12594.4	12611.6	12538.8	12957.5	13934.5
	DynamicJava	46689.9	45831.1	46802.2	47629.9	46623.6	46967.6	48846.0	47800.0	51495.8
	ImageJ	13620.5	13198.3	13574.2	13974.0	15923.0	15249.5	15624.4	14094.7	14033.5
	Rhino	13724.4	14813.0	14269.6	15688.7	14976.4	15996.8	15502.1	15749.5	15642.2

		0.10	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**
φ (DR)	*AutoFocus*	11966.2	**11700.4**	11932.1	12154.0	11868.6	11911.4	12021.0	11902.6	11865.9
	DynamicJava	47906.6	47848.9	46802.2	**46412.3**	46688.8	47295.8	47286.9	46780.8	48219.5
	ImageJ	14025.6	12855.3	13574.2	13479.9	13571.3	13076.8	12913.9	13068.5	**12837.2**
	Rhino	14667.1	14775.2	**14269.6**	15286.1	14404.9	14596.9	14668.6	14894.2	14617.0

		1	**4**	**8**	**10**	**20**	**50**
Q_H (HC)	*AutoFocus*	**11755.7**	11967.4	12013.7	11932.1	12897.3	12510.6
	DynamicJava	46747.3	46223.9	45508.7	46802.2	46258.4	**45141.6**
	ImageJ	14400.8	13365.5	14318.9	13574.2	13570.2	**12874.0**
	Rhino	14826.7	14942.4	14551.2	**14269.6**	14855.2	15357.2

		5	**10**	**15**	**20**	**50**	**100**
τ_{max} (PUB)	*AutoFocus*	12028.3	11932.1	12366.4	11564.0	12077.8	**11510.7**
	DynamicJava	48428.8	46802.2	48125.4	46293.4	45659.1	**45466.0**
	ImageJ	14667.2	13574.2	**13075.5**	13077.6	13642.6	13103.8
	Rhino	15371.3	14269.6	14688.5	14373.7	**13764.8**	13974.5

ABBREVIATIONS

PIF – Pheromone Importance Factor controls the weight of pheromone.
HIF – Heuristic Importance Factor controls the weight of heuristic information.
ER – Evaporation rate determines the evaporating rate of pheromone at each update.
DR – Decay rate determines the decaying rate of pheromone after every computation.
HC – Heuristic Constant is used to update the concentration of pheromone.
PUB – Pheromone Upper Bound controls the upper limit of pheromone.

Table 7: Impact of the different configurations on the execution time for four benchmark programs. The lowest execution time in each category has been highlighted.

5.2.3. Heuristic importance factor (HIF)

This factor controls the importance of the heuristic information. The heuristic information is useful for initial search and exploration of alternative routes when the environment has been changed. In this chapter, the heuristic criterion was primarily based on the logical distance of two nodes on the virtual space. Theoretically, a greater β value will result in a shorter length until such a length is saturated as illustrated in Figure 23.

Figure 23 shows that the path lengths of four benchmarks were failing sharply from 1.5 to 10. With benchmarks AutoFocus, DynamicJava and Rhino, the lines gradually became flat as β grew to 50. However, the lookup length for ImageJ climbed above 4.5 at 20 before the length went flat towards 50.

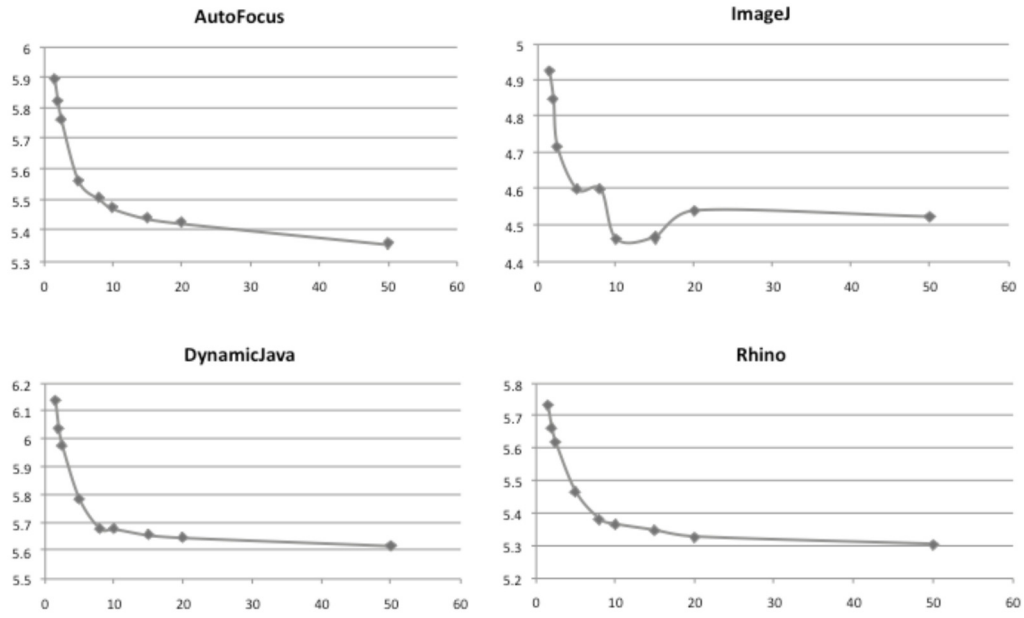

Figure 23: The path length versus the heuristic importance factor for four benchmarks.

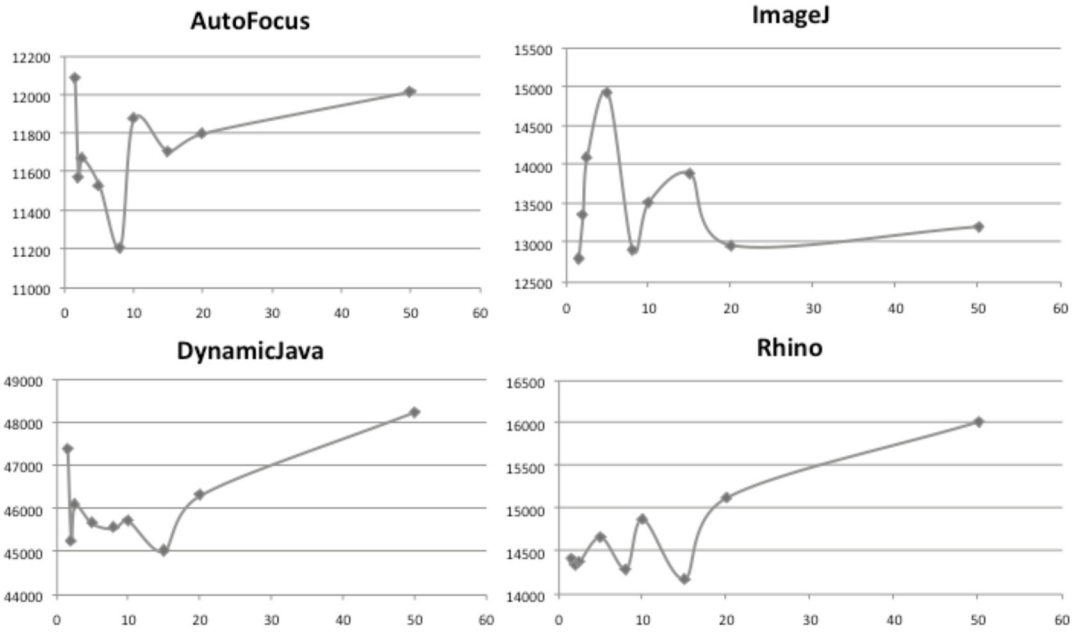

Figure 24: The heuristic importance factor versus the simulation completion time for four benchmarks.

With respect to the completion time, four benchmarks showed distinguish results as illustrated in Figure 24. With the AutoFocus, the lowest completion time occurred when β was around 8. On the other hand, the lowest time for the DynamicJava and the Rhino was near β = 15. The last benchmark ImageJ showed great frustrations in the completion time at the low β values. With ImageJ, the lowest time and the second lowest time measured respectively at β = 1.5 and β = 8.

Tests results again demonstrated that the length of the lookup paths was not linearly related to the locating efficiency. The path length is indeed one of many factors that could affect the locating efficiency.

5.2.4. Evaporation rate (ER)

The evaporation rate determines the dissolving rate of the pheromones. Theoretically speaking, the higher rate leads to the faster evaporation process and consecutively it reduces the system's capability to utilise existing solutions.

In Figure 25, the evaporation rate showed a negative influence to the average lookup path lengths, since the path lengths became longer when ρ was increased from 0.1 to 0.9.

The ImageJ benchmark is, however, an exception, where the path length first decreased from the highest point, 4.996 to the lowest point, 4.747 and then reached 4.97 at $\rho = 0.9$. As discussed earlier, the pheromones are beneficial to the great deal of temporal calls found in the ImageJ. Thus, the right evaporation setting can effectively reduce the average length for programs like the ImageJ. In general, faster evaporation causes similar queries to travel on different routes unnecessarily or slower evaporation discourages dislike queries to look for new alternatives. Furthermore, the highest point on the plot of the lookup paths was found at $\rho = 0.2$ and the lowest point at $\rho = 0.5$. Interestingly, this was inverse to the completion time; the peak of the completion time occurred at 0.5 and the deepest point at 0.2 as depicted in Figure 26.

Finally, when the evaporating rate is fast, the search process will not be able to take advantage of previously established solutions but have to rely on the heuristic information. Under such a condition, the algorithm is similar to a stochastic multigreedy algorithm [136].

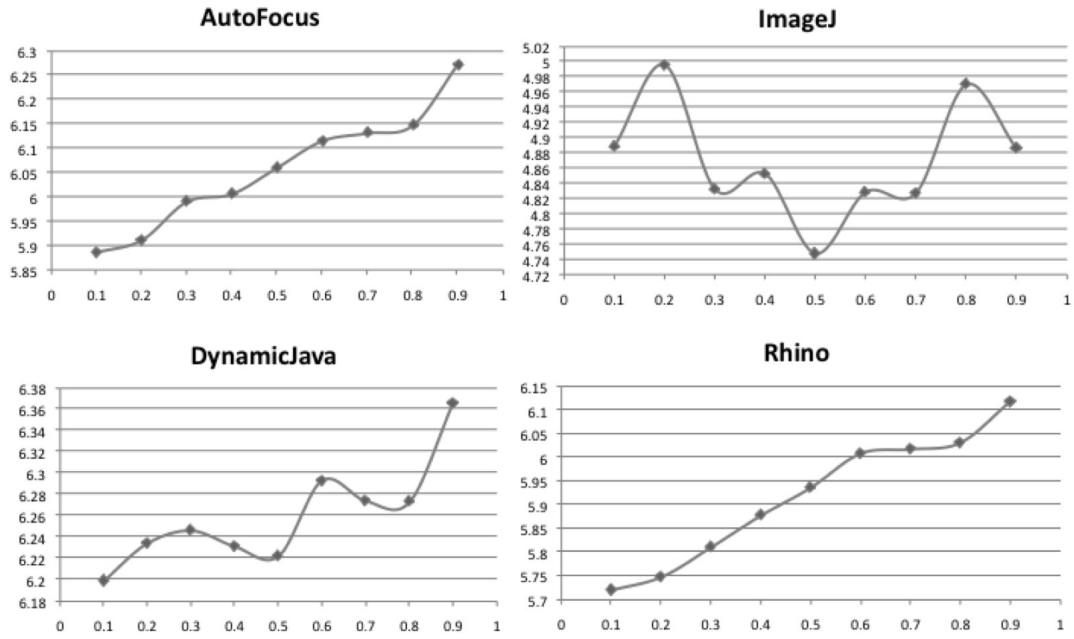

Figure 25: The evaporation rate versus the lookup path length for four benchmarks.

5.2.5. Decay rate (DR)

The strength of a path's pheromone is immediately diminished according to the decay rate after such a path is selected for passing the current query. This can encourage parallel queries passing the same node to explore alternative paths during the same search cycle.

The simulation results showed that the best φ values for the lookup length were diverse in all benchmark programs. For the execution time, the best φ values occurred at the range of 0.2 and 0.4 apart from the ImageJ benchmark. For ImageJ, its best φ value was 0.9.

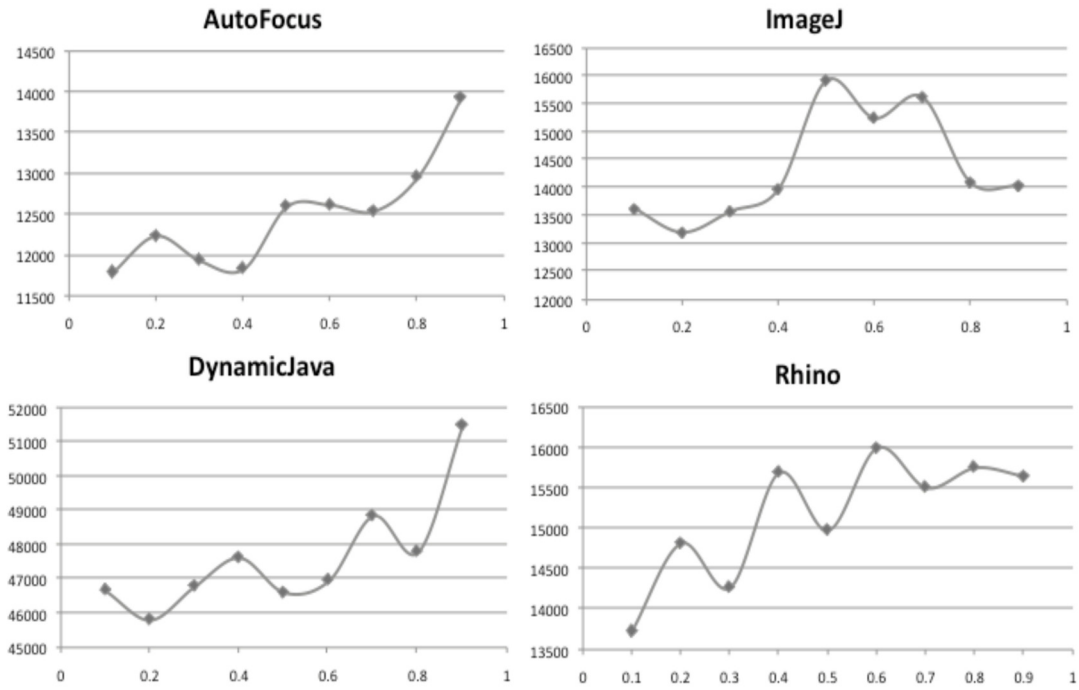

Figure 26: The evaporation rate versus the simulation completion time.

5.2.6. Heuristic constant (HC)

This constant is used to control the quantity in updating the concentration of a path's pheromone. Generally, the growth of the concentration is proportional to the amount of the Q_H constant.

Figure 27 illustrates the average locating length as the function of the heuristic constant. The minimum points for four benchmarks were found when the Q_H values were no greater than 20. However, the locating length of the DynamicJava began to decrease when Q_H was increased from 20 to 50.

On the other hand, the completion time frustrated when Q_H was smaller than 10. The plots of the ImageJ and the DynamicJava slide sharply to the lowest points afterwards, while the plot of the Rhino raises quickly after passing 20.

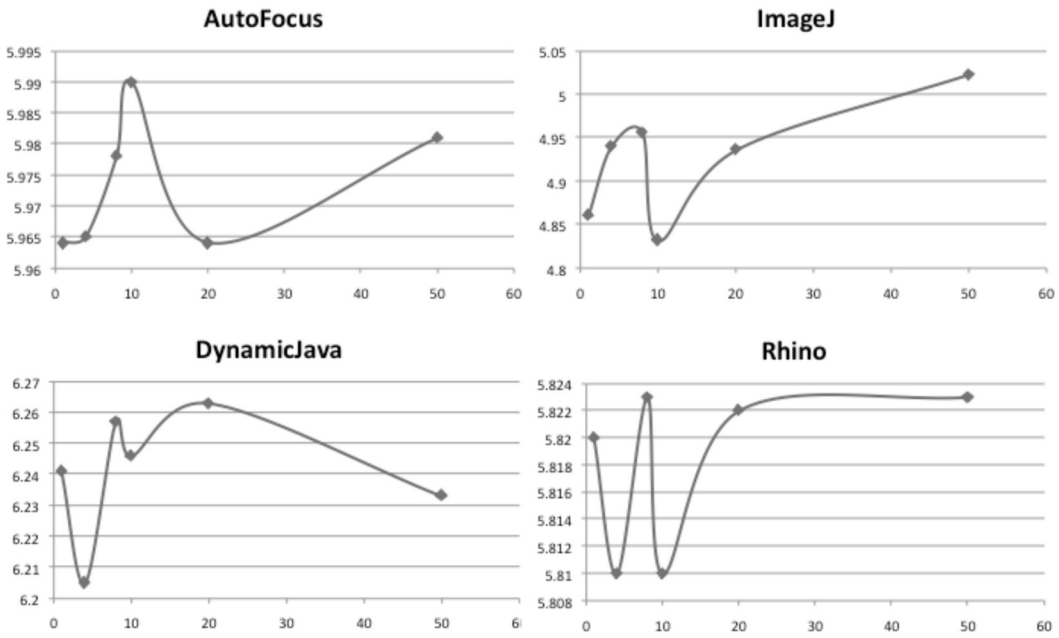

Figure 27: The heuristic constant versus the average locating path length.

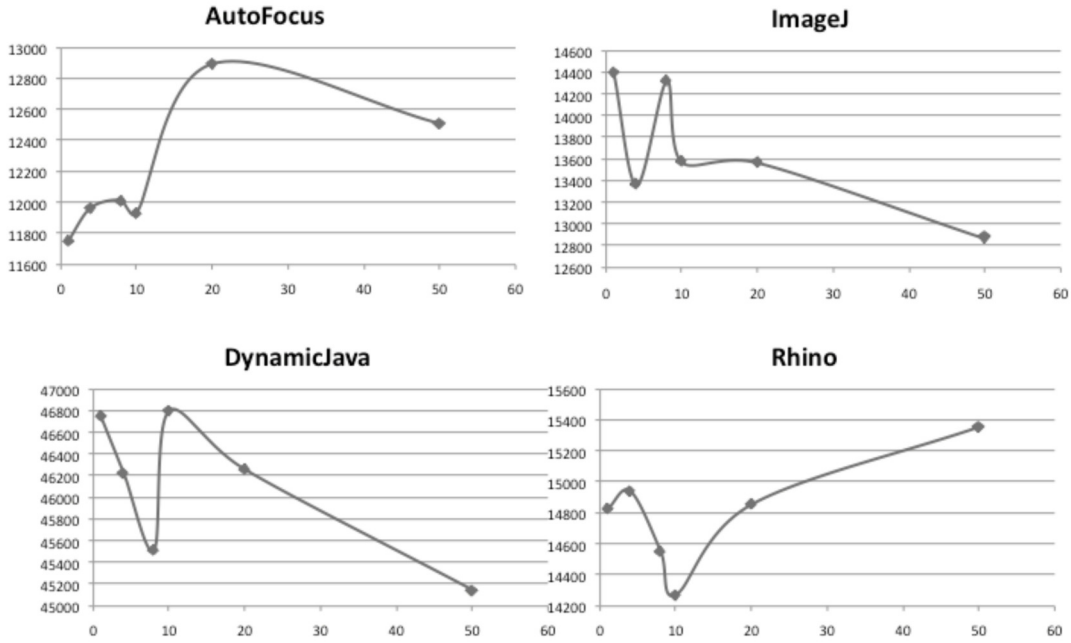

Figure 28: The heuristic constant versus the simulation completion time.

5.2.7. Pheromone upper bound (PUB)

This parameter restricts the growth in the concentration of the pheromone as it controls the impact of the pheromone on the probability computation.

The benchmarks AutoFocus and Rhino showed for τ_{max} a monotonic decrease of the lookup length in Figure 29. However, the benchmarks DynamicJava and ImageJ showed that the lookup length was in reversely proportional to the τ_{max} value.

For the execution time, optimum τ_{max} numbers were found differently for each benchmark as illustrated in Figure 30. In general, the greater values gave the better performance.

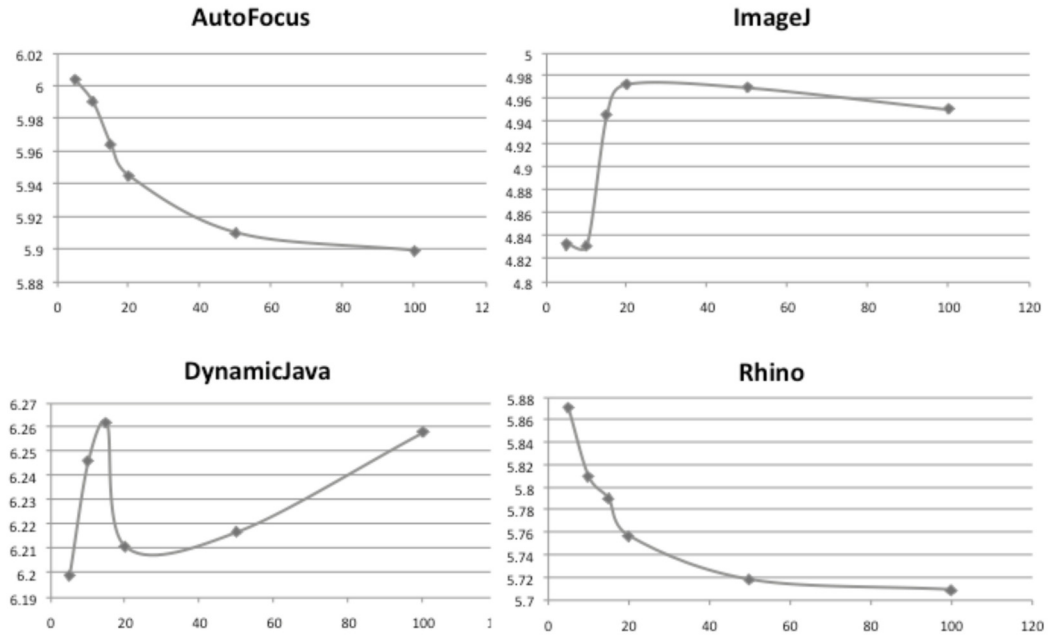

Figure 29: The pheromone upper bound versus the average locating length.

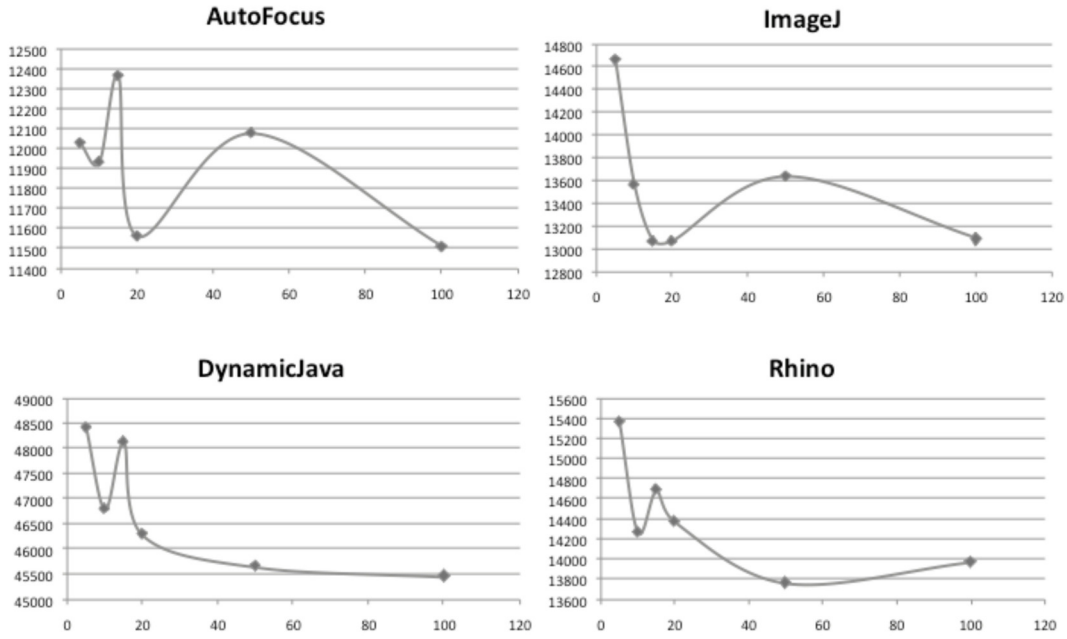

Figure 30: The pheromone upper bound versus the simulation completion time.

5.2.8. A summary of the test results

The tests showed that manipulating the parameters of the ACO algorithm could alter the search performance. However, the simulation study indicated that the impact of some parameters was significantly diverse in the four benchmark programs. This is due to the differences in the characteristics of object communication amongst programs. Furthermore, the tests showed that the lookup length and the execution time had quantitatively different behaviours as the result of parameters. This implies that the lookup length is not necessarily the factor that most influences the execution efficiency. Hence, it is desired to incorporate other factors in order to optimise for the execution efficiency. A detailed study of the optimisation techniques is presented in Chapter 6.

Software that embeds with a large number of temporal message passing was sensitive to the pheromone importance factor (α). A sender can send multiple messages to the same recipient within a short period. Those messages may be delivered based on symbolic names as the first reply message that includes the recipient's physical address, may arrive at the sender later because of the geographical distance between the sender and

the recipient. To improve the efficiency of routing the temporal messages, ARMS adopts pheromones to encourage all messages targeted the same destination to follow the best path that is known so far. It showed that the increase of the pheromone importance factor in the path selection process led to a shorter forwarding path for programs with a great deal of temporal calls, such as the ImageJ program. However, a very large value of α could damage the performance of the object locating for programs with a small portion of temporal calls. The simulation study suggests that it is important to maintain the pheromone factor under 20.

The simulation results showed that the values of the heuristic importance factor (β) had a significant impact on the lookup length. Generally, the length is inversely proportional to the value of β. The tests showed that the value of the heuristic importance factor is desired to be higher than the value of the pheromone importance factor as the heuristic information is vital for the beginning of the search. However, the outcome depends on the choice of the heuristic information. In this study, the heuristic information is selected mainly based on the logical distance between two nodes on the virtual space. Consequently, a higher value of β can lead to a shorter lookup path.

All benchmark programs showed close behaviours with respect to the evaporating rate (ρ). The results demonstrated that a slow evaporating rate was desired because it allows query messages to exploit the good solutions that have been discovered previously.

Lastly, it is worth noting that – as observed by Dorigo [136] – for high values of α and not too high values of β the algorithm converges very quickly to the unipath behaviour without finding very good solutions.

5.3. A Study of Lookup Scalability

5.3.1. State of forwarding table

This sector reports four sets of tests for study of the scalability of ARMS: the size of the forwarding table, the amount of parallel query messages, the network size, and the object populations. The results were also compared to the Chord counterpart.

Firstly, the randomised network of ARMS allows each peer's routing state to expand to support the proximate routing selection and to reduce the network diameter. This, however, can result in a significant growth of the routing state compared to the Chord's routing state. The comparison result is depicted in Figure 31. Because it can cause unbalanced workloads in the network, ARMS makes use of the phenomenon to achieve the optimal size of routing tables. In ARMS, any fingers that have low pheromone values will be removed. Thus, it can slow down the growth of routing states. Figure 31 also showed that Rhino and AutoFocus had bigger routing states because they produced more objects during the execution. However, the size of the Chord's routing tables remained consistent despite different object populations embedded in the benchmarks.

Figure 32 illustrates the progressing average routing state for every benchmark on the normalised simulation time. Obviously, the fastest increase in the size of the routing table happened between 30% and 70% of the execution time due to the growth of the object populations. The size of the routing table reached a matured point approaching to the end of the execution. With the AutoFocus and the Rhino, the matured point is closed to the asymptotically optimal routing size – $\log_2(900) \cong 9.8$.

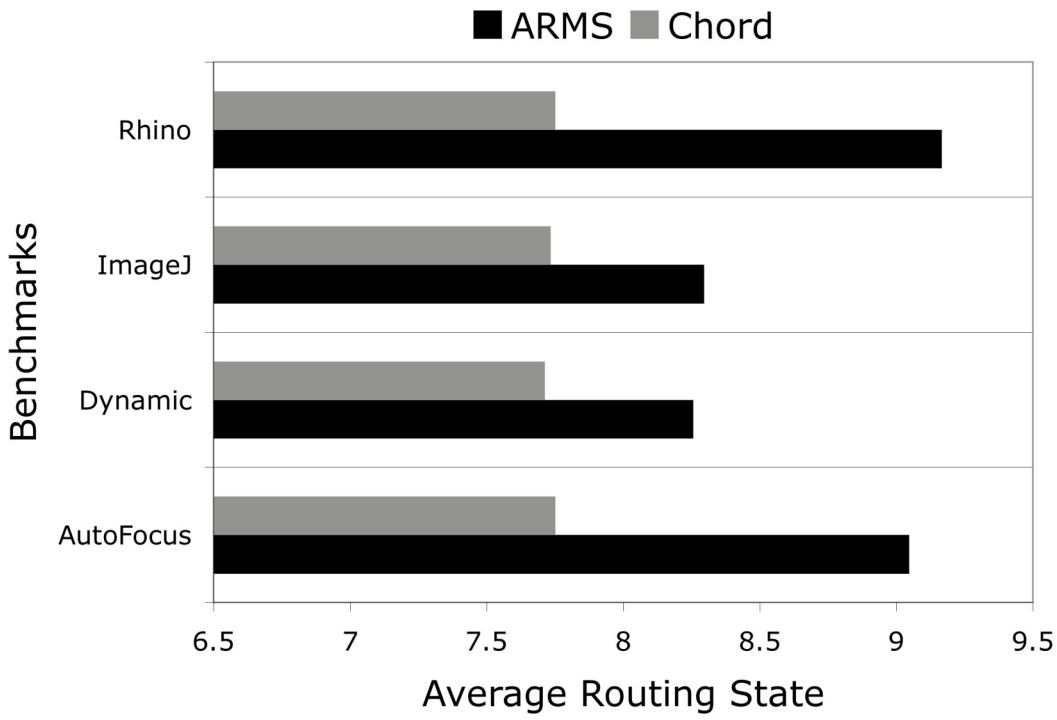

Figure 31: A comparison of the routing state between ARMS and the Chord for each benchmark.

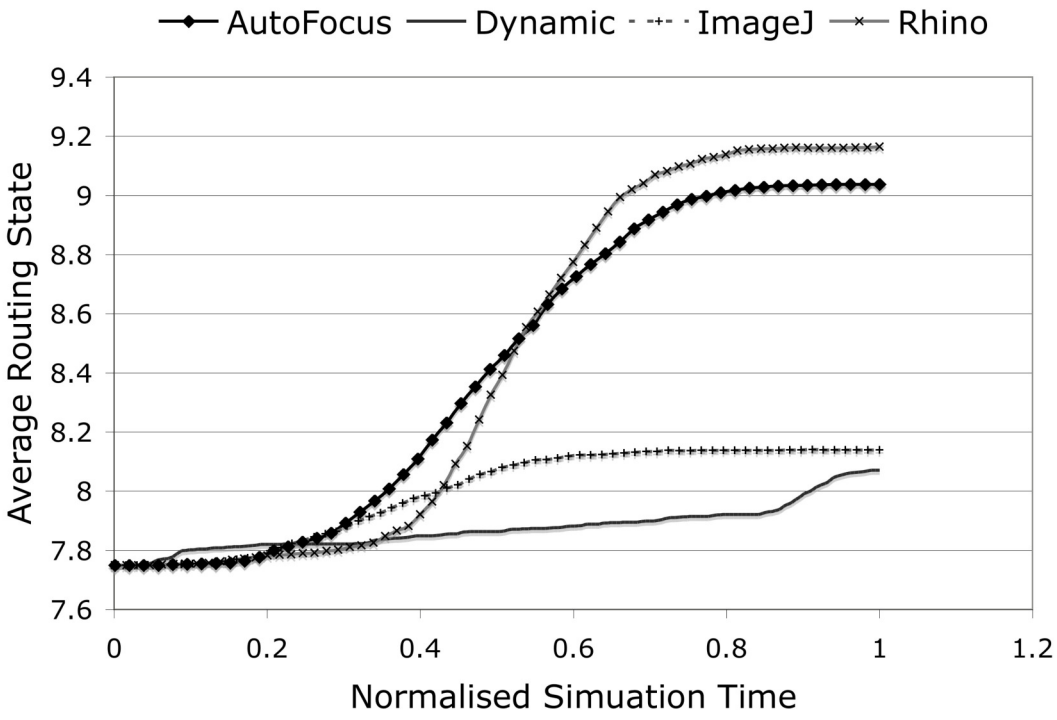

Figure 32: An illustration of the progressing routing state in ARMS for each benchmark.

5.3.2. Size of parallel query messages

In ARMS, a requesting node generates a number of query messages to be forwarded to neighbours as analogue to the creation of artificial ants in the local ACO algorithm. In theory, the size of the initial query messages has direct impact on the efficiency of the execution as more messages can improve the chance of discovering a quicker route.

In the tests, a node produced the quantity of the parallel messages as the fraction of the chosen neighbours, ranging from 10% to 100%. Furthermore, the values of the ACO parameters were chosen based on the previous tests: $\alpha = 10$, $\beta = 50$, $\rho = 0.1$, $\varphi = 0.8$, $Q_H = 10$, $\tau_{min} = 1$ and $\tau_{max} = 50$.

The differences of four benchmarks in terms of the lookup length and the execution time are demonstrated in Figure 33 and Figure 34, using the Chord as the benchmark. As depicted in Figure 33, the increase of the initial query messages led to slightly longer lookup lengths. As more messages were sent, the likelihood that two messages arrived at the same node was increased. Due to the decay coefficient, the second message arrived at the node was encouraged to select an alternative path that may require travelling with a longer length.

However, as illustrated in Figure 34, the behaviour of the execution time was dissimilar. For benchmarks DynamicJava and Rhino, the increase of the initial query messages led to more efficient execution. This is because sending more messages can reduce the time spent on finding a node that knows the current address of the requested object. However, the execution time is not linearly proportional to the size of the initial messages as shown in the figure. While sending more messages enhanced the system's exploration ability, the excessively large amount of messages caused network

congestions. Thus, there is a trade-off between the exploration need and the communication efficiency.

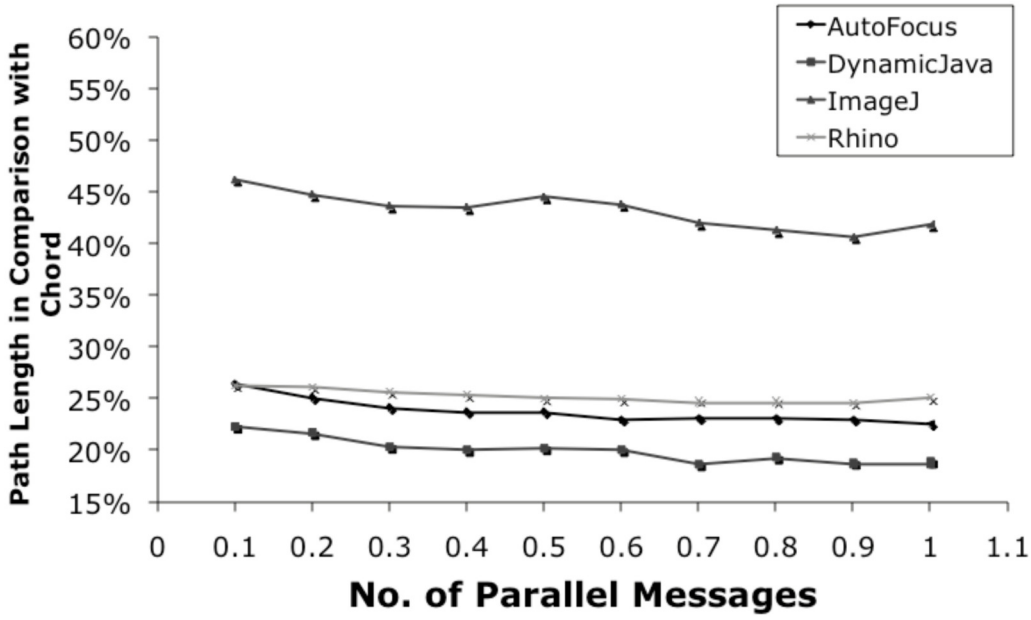

Figure 33: A demonstration of the lookup length as the function of the amount of the initial query messages. The initial messages were changing from 10% to 100% of the size of candidate neighbours.

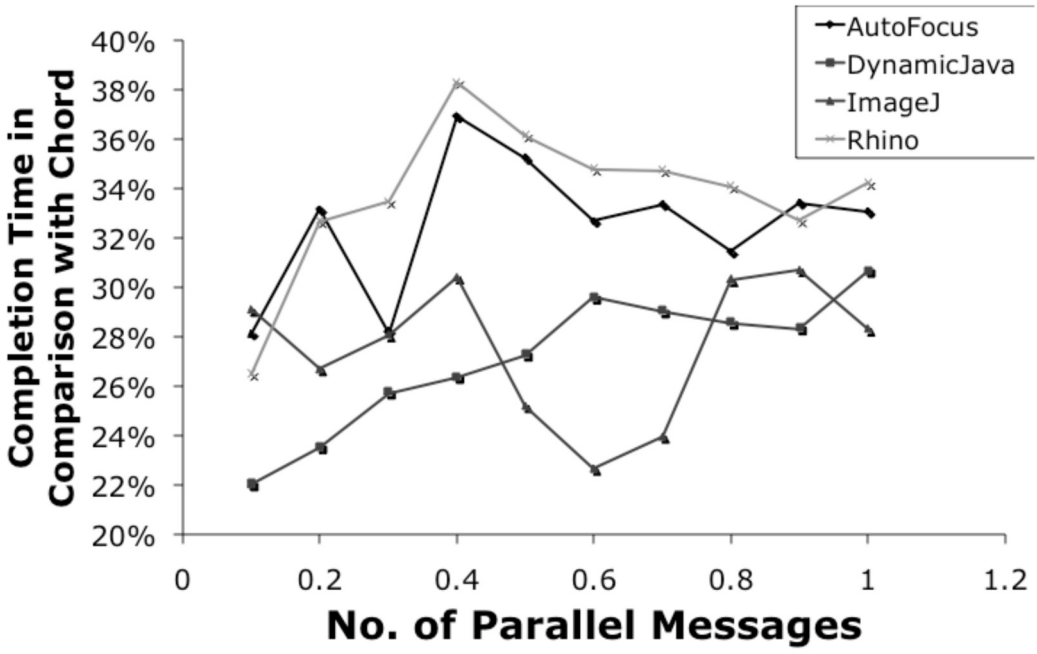

Figure 34: A demonstration of the execution time as the function of the amount of initial query messages The initial messages were changing from 10% to 100% of the size of candidate neighbours.

5.3.3. Network size

The size of a distributed system needs to be upgraded continually to facilitate the increasing complexity of software. Hence, it is vital to design a locating scheme with a good scaling quality for such a system. This section studied the scaling property of ARMS with respect to the size of nodes in the system, termed the network size. In the tests, the size configurations of the network size were simulated, respectively, 169, 256, 484, 1024, 2116 and 4096. Two benchmark programs DynamicJava and ImageJ were selected for simulation since results were similar in all benchmarks. Again, the same ACO parameters were applied and several simulation studies for each configuration were performed in order to obtain the mean values.

Figure 35 and Figure 36 show an exponential growth of the lookup length as the increase of the network size. It is noted that the figures have been drawn with the logarithm setting. Furthermore, points above the line represent the maximum length in each category while points below the line represent the minimum length in each category. Clearly, both ARMS and the Chord showed a good scaling capability regarding the network size. However, the length of the lookup path in ARMS grew slower than the Chord did with the increasing network size. On average, ARMS required six nodes in the query forwarding on the network of 3364 nodes, while the Chord required eight nodes in the query forwarding on the same size. This is essentially due to the small radius of the randomised overlay and the adaptive locating protocol.

On the other hand, the plots of the execution time versus the network size are illustrated in Figure 36 and Figure 37 for benchmark programs DynamicJava and ImageJ. Two models showed approximately the similar quantity of the execution time when the

number of the nodes is less than 256. However, the execution time of the Chord rose more rapidly than the time of ARMS for a larger network size.

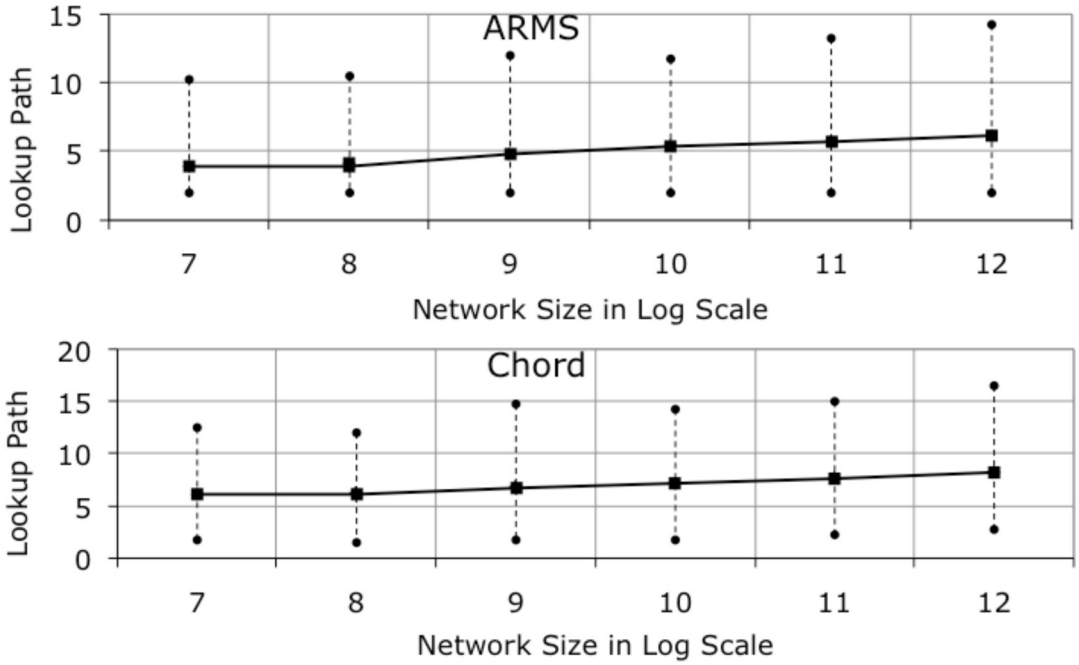

Figure 35: A logarithmic plot of the length of the lookup path as the function of the network size for DynamicJava. Each column shows three values from the top to the bottom: the longest lookup path, the expected lookup path and the shortest lookup path at each network size.

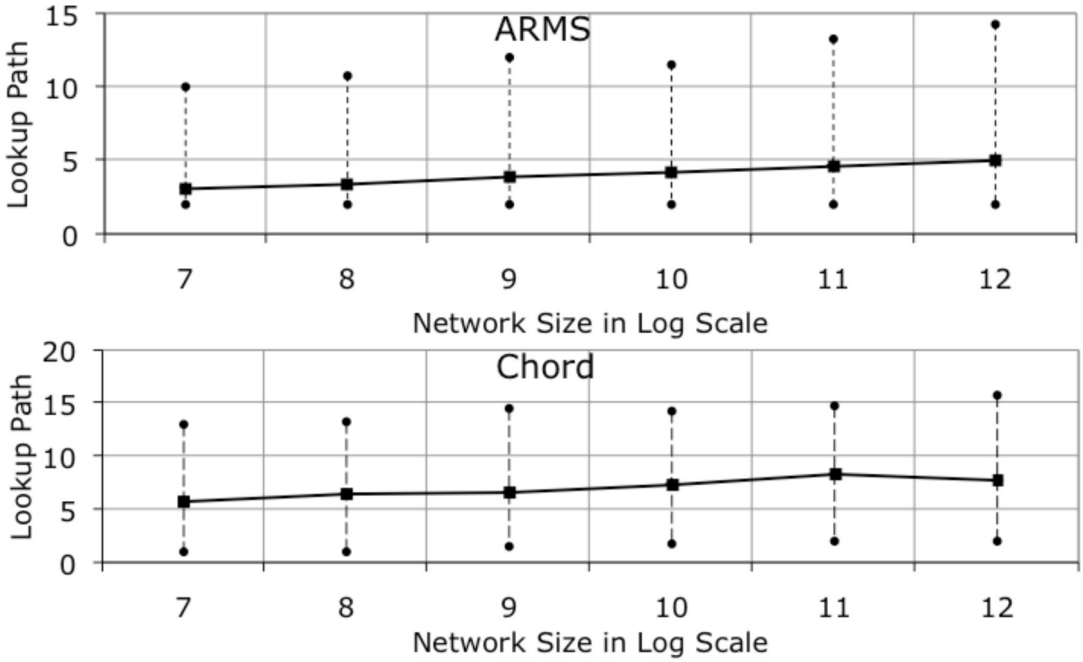

Figure 36: A logarithmic plot of the length of the lookup length as the function of the network size for ImageJ. Each column shows three values from the top to the bottom: the longest lookup path, the expected lookup path and the shortest lookup path at each network size.

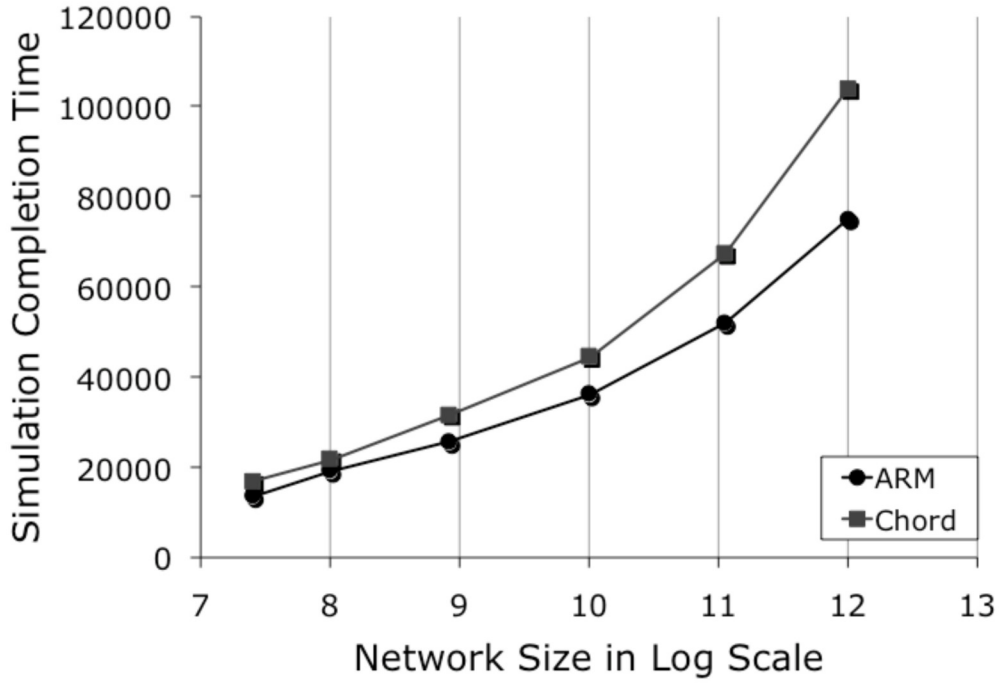

Figure 37: A plot of the length of the execution time as the function of the network size for DynamicJava.

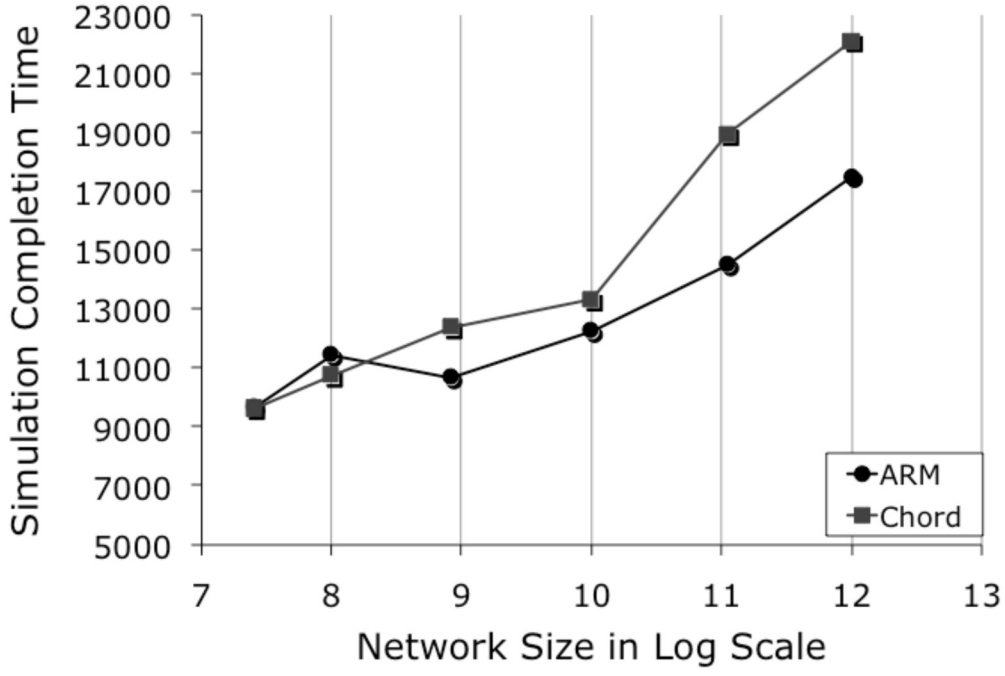

Figure 38: A plot of the length of the execution time as the function of the network size for ImageJ.

5.3.4. Object population

Next, the performance of the system versus the object population under different conditions was tested. The simulation was conducted using two benchmark programs – ImageJ and Rhino, with various sizes of the object populations. Three sets of the object populations for ImageJ were 48, 91 and 144 while other three sets for Rhino were 531, 896 and 1088.

As shown in Figure 39 and Figure 40, the lookup length tended to be independent to the size of the object population. However, the execution time had a linearly relationship with the population size. In general, more objects resulted in more messages created in the system. As a consequence, the execution time went up as the increase of the object population as shown in Figure 41 and Figure 42.

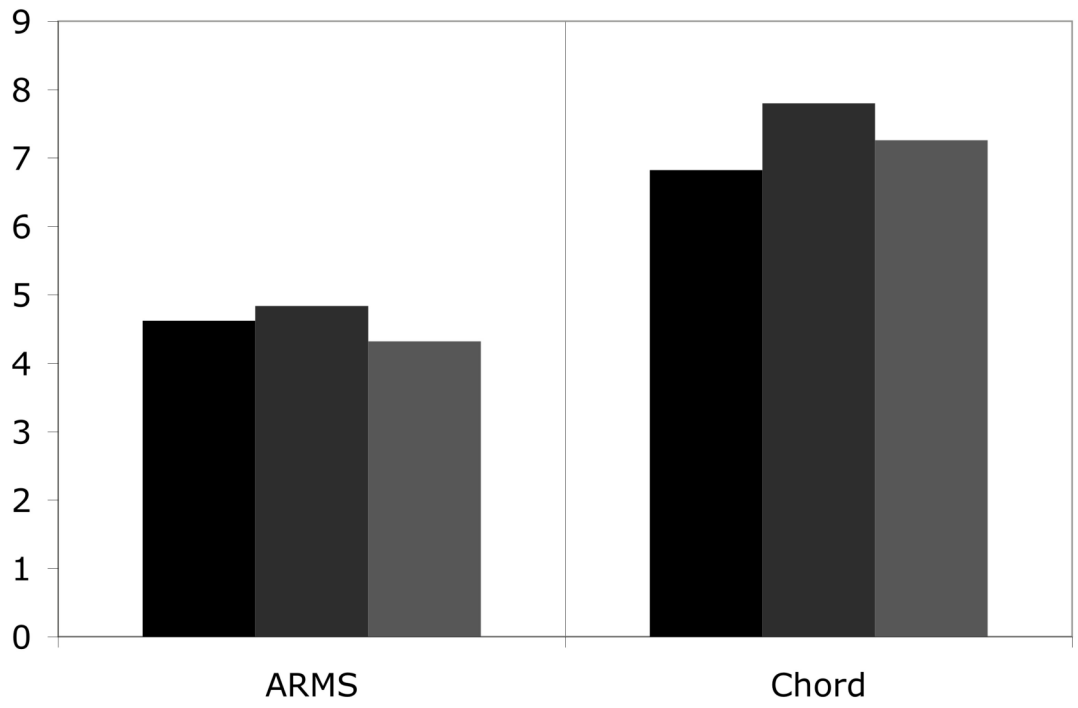

Figure 39: A plot of the lookup length as simulating different sizes of the object populations using the benchmark ImageJ.

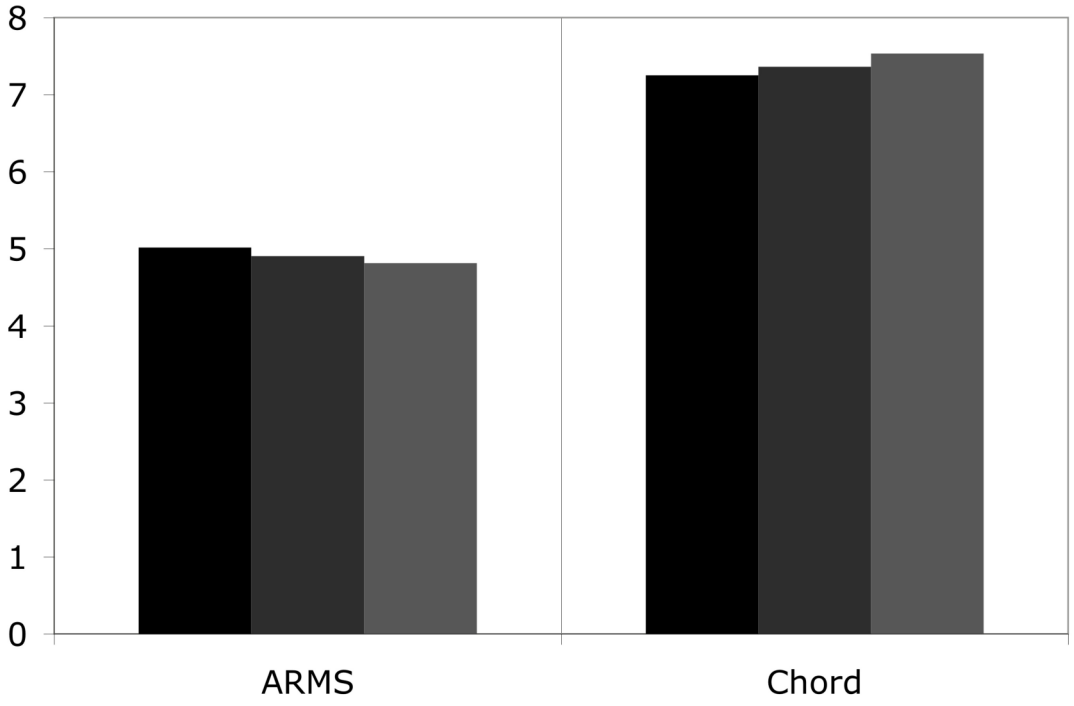

Figure 40: A plot of the lookup length as simulating different sizes of the object populations using the benchmark Rhino.

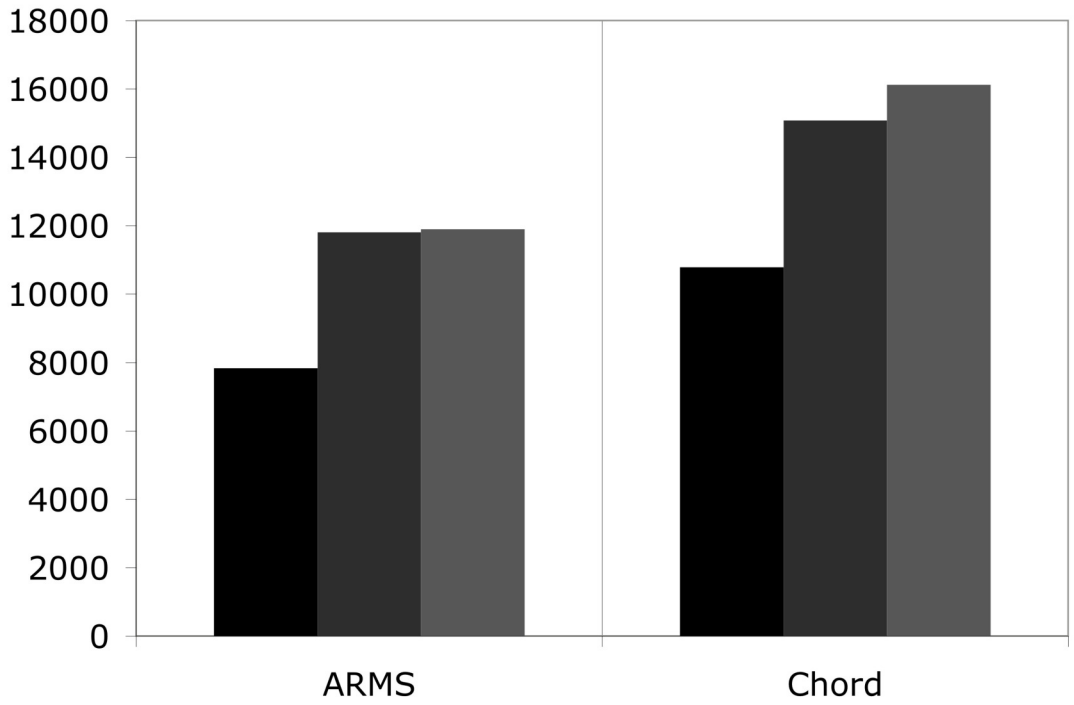

Figure 41: A plot of the execution time as simulating different sizes of the object populations using the benchmark ImageJ.

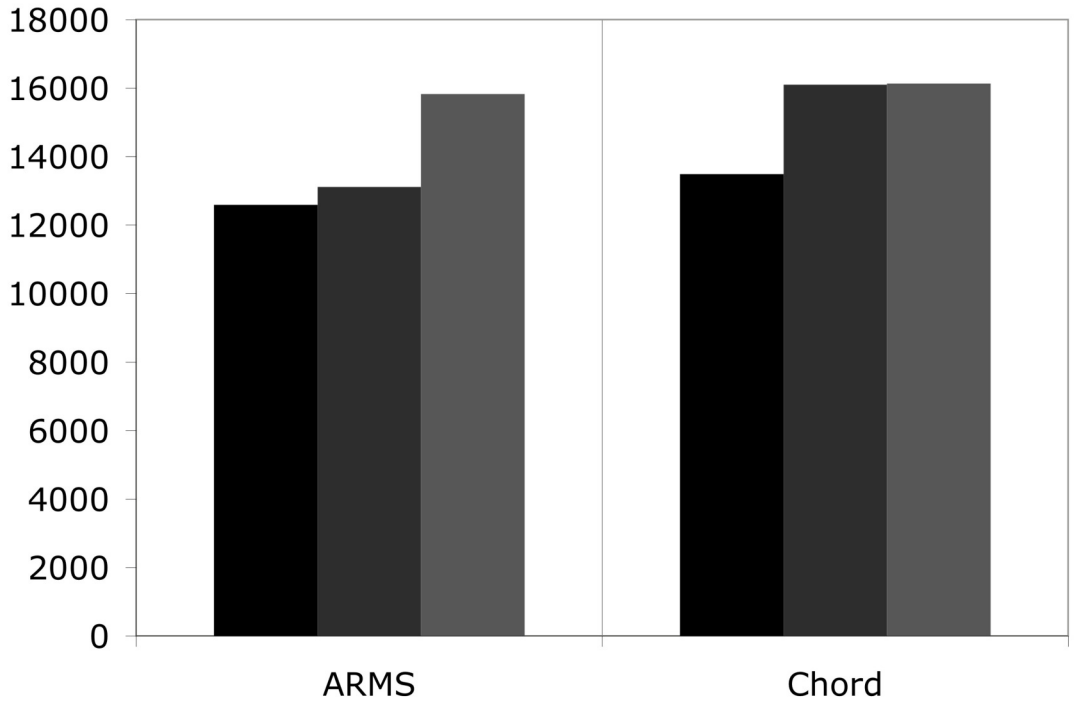

Figure 42: A plot of the execution time as simulating different sizes of the object populations using the benchmark Rhino.

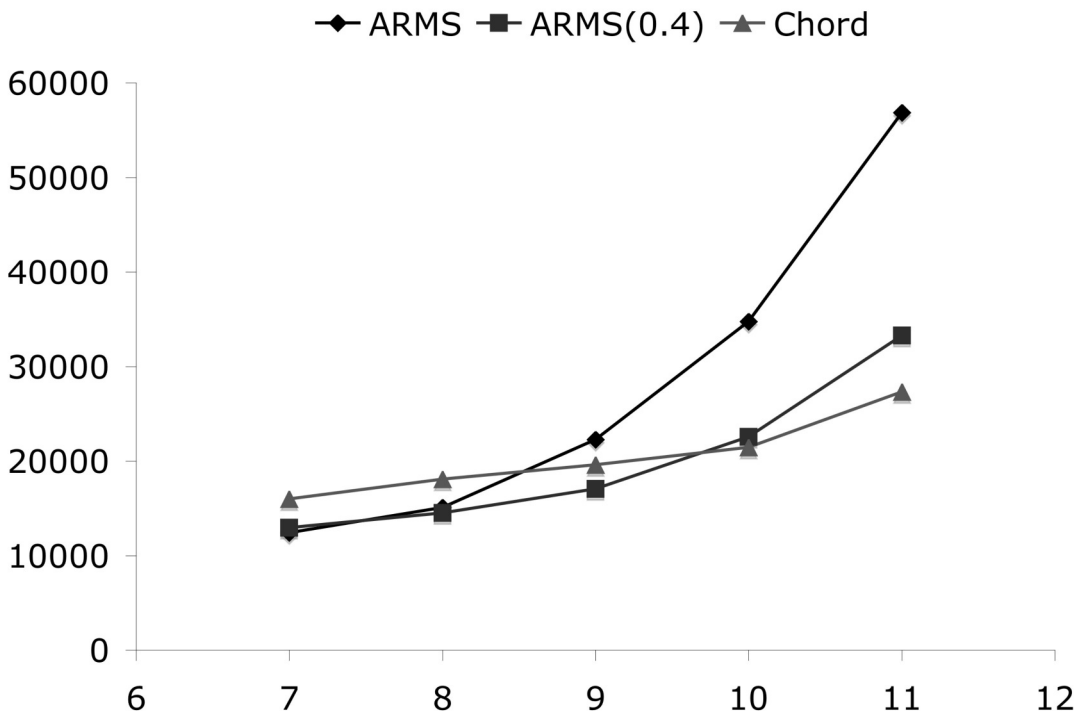

Figure 43: A plot of the execution time as simulating different sizes of the object populations generated by a random model.

Figure 43 demonstrates the execution time with different configurations. Five separate sets of tests were conducted using a random model. Unlike the network traffic produced by the benchmark programs, the random model generates uniformly distributed message passing between objects. The tested object populations were 128, 256, 512, 1024 and 2048.

While tests showed that the performance of ARMS was better than the Chord in the small size of the object population, the efficiency of ARMS was sensitive to the growth of the object population because of the use of multicasting in the query forwarding. The multicasting approach can potentially saturate the network with a large number of messages and hence extra messages due to the growth of objects will likely cause serious link congestions. Consequently, the execution time of ARMS increased exponentially as the increase of the object population when the sender used its entire neighbour links to propagate the query messages. However, the problem was overcome by cutting down the number of the links to 40% of the neighbour links. This substantially slowed the increase of the execution time quite dramatically. The study has shown that using 10% – 40% of the neighbour links in the multicasting could achieve the efficient name resolution yet retain good scalability.

5.4. Implementation on Different Network Structures

The performance of ARMS implemented on a cube network has been examined. Four benchmark programs were tested on the cube network that consisted of 1000 nodes. The tests showed that the lookup lengths of ARMS were similar between two types of networks. This is due to the use of a virtual topology as the platform for search. Such a topology is invariant to the underlying physical structure and thus the lookup path is independent to the actual network structure.

However, the test results showed that the execution time was significantly different in the two networks as depicted Figure 44 and Figure 45. The execution time simulated in the cube network was much lower than the execution time in the cluster network. The phenomenon can be explained via Table 8. The table summarises the total quantity of query messages sent, the accumulated travelling distance by query messages and the average travelling distance per message. It is clear that the distinction in the total number of query messages was insignificant between two networks.

On the other hand, the accumulated travelling distance was substantially different as the messages can take advantage of the larger number of node connections and the higher dimensional network structure. Consequently, the average travelling distance per message was lower in the cube network. The results suggest that there is advantageous to optimise the query forwarding for the distance travelled in the underlying physical network.

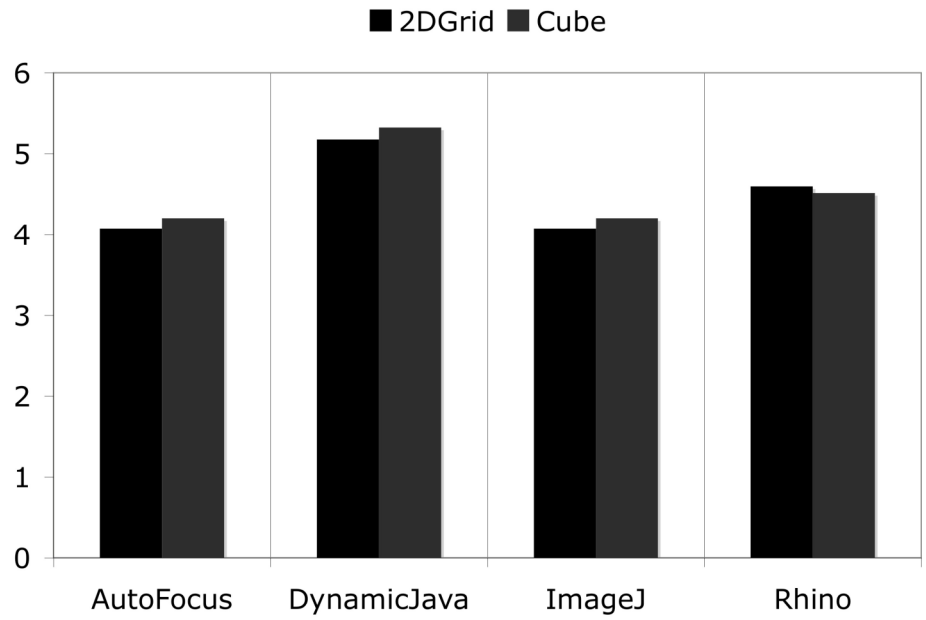

Figure 44: An illustration of the lookup length as simulating ARMS on two types of network structures, 2D grid and cube.

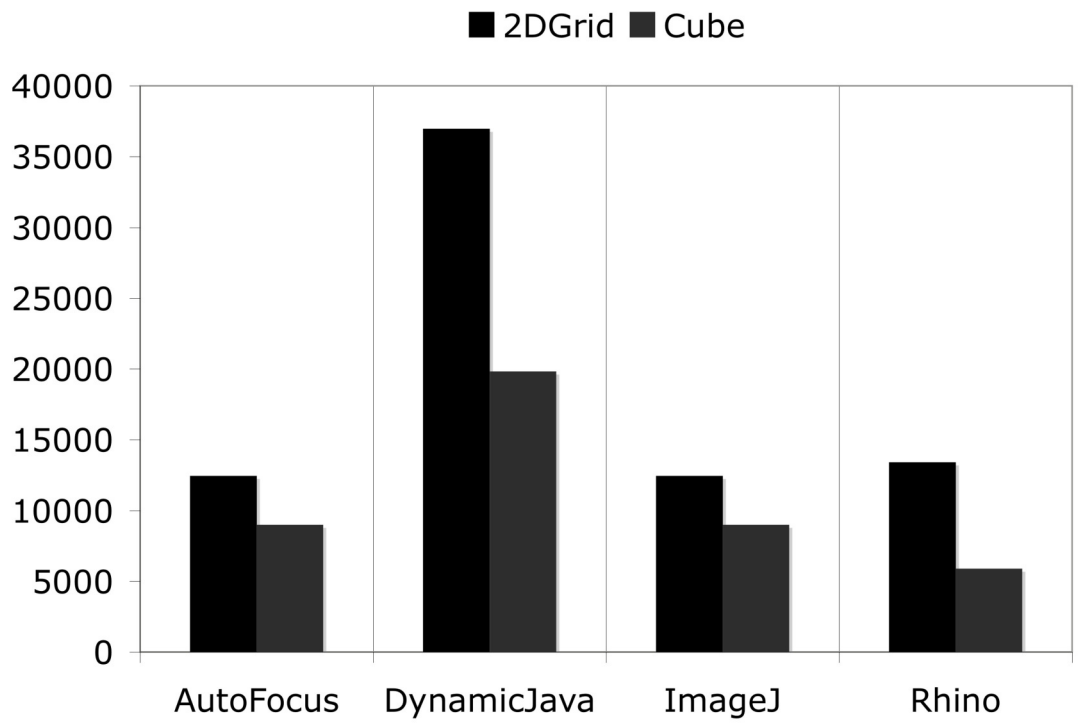

Figure 45: An illustration of the execution time as simulating ARMS on two types of network structures, 2D grid and cube.

Parameter	AutoFocus		DynamicJava		ImageJ		Rhino	
	2D Grid	**Cube**	**2D Grid**	**Cube**	**2D Grid**	**Cube**	**2D Grid**	**Cube**
Total queries	4704	4538	1486	1476	4704	4538	13817	13562
Distance	408062	127097	160128	51645	408062	127097	1346901	409127
Avg. distance	86.69	28.00	107.75	34.94	86.69	28.00	97.49	30.17

Table 8: A summary of the test results for the total amount of query messages, the total distance travelling by the messages, and the average travelling distance per message.

6. Impacts of Constructive Heuristic Information on the Locating Efficiency

6.1. Path Exploration for Improving Object Locating Performance

In the last chapter, the effect of the pheromone values on the performance of object search was examined. The characteristic of the pheromones allows query messages to exploit an efficient route that is constructed in the past. In addition, any link whose pheromone is below a threshold will be removed from the virtual network in order to optimise the network complexity. However, a path selection rule is vital to ensure that an efficiency solution can be converged quickly at the beginning of a route construction. Otherwise, when the condition of a link is changed, the selection rule is required for discovering a better alternative. Consequently, the ability to explore paths is as critical as the exploitation ability to the query performance. In ARMS, the exploration ability is governed by constructive heuristic information. This chapter examines four kinds of heuristic information as optimisation strategies for strengthening the ARMS capability of the dynamic path exploration.

Firstly, the organisation of the virtual space should be transparent to the structure of the underlying physical network so that the physical layer can be changed independently without affecting the logical layer. Therefore, there is little connection between the topology of a virtual space and its underlying network structure. That is, two nodes are neighbouring on the virtual space might be located far apart from each other in the physical layer. This is known as the proximity problem. Hence, the first type of heuristic information is based on the *proximity of neighbouring peers* in query routing.

The second strategy has considered the *load balancing* issue in the system. The motivation here is to reduce the number of hotspots in the system by balancing the number of processing queries by each node. In ARMS, an efficient route will attract more query messages to use that route. This can increase the workloads of the links and the peers along the route and will subsequently cause traffic congestions. Therefore, it is ideal to distribute some query messages to other routes that may be longer to reduce the risk of congestions. This strategy compensates load balancing with a long search path travelled in the virtual space and/or the physical network.

In Chapter 4, the profiling study showed that the amount of messages sent or received by every object was not uniformly distributed. Generally, the objects exchanged more messages would have more knowledge of the locations of other objects than those involved in fewer messages. This suggests that the lookup efficiency can be enhanced possibly by searching via nodes contained objects that have higher degree of connectivity. As a result, the third strategy for optimisation is based on the *degree of connectivity*. However, the use of the degree of connectivity will have significant impact on the load balancing of the system.

Finally, the *spatial property* of message passing shows potential for constructing heuristic information. Similarly, a query message should travel towards nodes contained objects that have programmatic relations with the requested object in the software.

For simulation, the default value of the ACO parameters was $\alpha = 2$, $\beta = 10$, $\rho = 0.1$, $\varphi = 0.8$, $Q_H = 10$, $\tau_{min} = 1$ and $\tau_{max} = 50$. Totally five dimensions of the networks were simulated for each configuration, respectively, 1156 nodes, 1600 nodes, 2116 nodes,

2704 nodes and 3364 nodes. The performance measures have been considered in this study include

- *Execution time* (ET) – measures the amount of the simulation time spent on executing object invocations that are contained in a trace file;

- *Distance per query* (DPQ) – measures the average distance for every query message travelled in the physical network;

- *Mean lookup length* (MLL) – measures the average lookup length for every query message on the virtual structure;

- *Average workload* (AWL) – measures the average number of the processed queries for each node; and

- *Workload standard deviation* (WSD) – measures the symmetry of the workloads crossing all nodes in the system.

6.2. Optimisation Strategies

6.2.1. Optimising for proximity

The incompatibility between the virtual structure and the network organisation could result in a long travelling distance by the query message. Therefore, the heuristic information of the ACO algorithm is modified for the minimum travelling distance. The heuristic information for the link C_{ij} that joins node i and node j is given by

$$\eta_{ij} = \frac{H_{ij}}{\sum_{C_{ik} \in N(S^P)} H_{ik}}, \text{ if } C_{ij} \in N(s^p) \tag{6.1}$$

where $N(s^p)$ is the set of unvisited neighbours, H_{ij} is the inverse of the Manhattan distance between node *i* and node *j*,

$$H_{ij} = \frac{1}{d_{ij}} \tag{6.2}$$

Instead of using the Manhattan distance, the link latency could be used as an alternative measurement. Furthermore, there are two ways to update the heuristic information: periodic updating or piggybacking in messages. Nonetheless, the piggybacking, where the update information is embedded into the reply message, was used in the simulation since it requires fewer messages than the periodic updating approach does.

Table 9 and Table 10 summarise the test results of the ARM_LD and the ARM_PR through simulating four benchmark programs, namely, AutoFocus, DynamicJava, ImageJ and Rhino. As illustrated in the tables, the value of MLL was much smaller in the ARM_LD than the ARM_PR. In contrast, the value of DPQ was lower in the ARM_PR than the ARM_LD as expected. Furthermore, giving the benchmark DynamicJava as an example, the test results are also presented in Figure 46 and Figure 47 for better illustration. Additionally, the Chord was also used for the benchmarking purpose. Clearly, the Chord had the highest values for the MLL and the DPQ among three schemes due to the limitation of its preliminarily determined path. The test results showed the flexibility of ARMS as the performance of the scheme can be adjusted

through the ACO parameters or the heuristic information based on the application of the system.

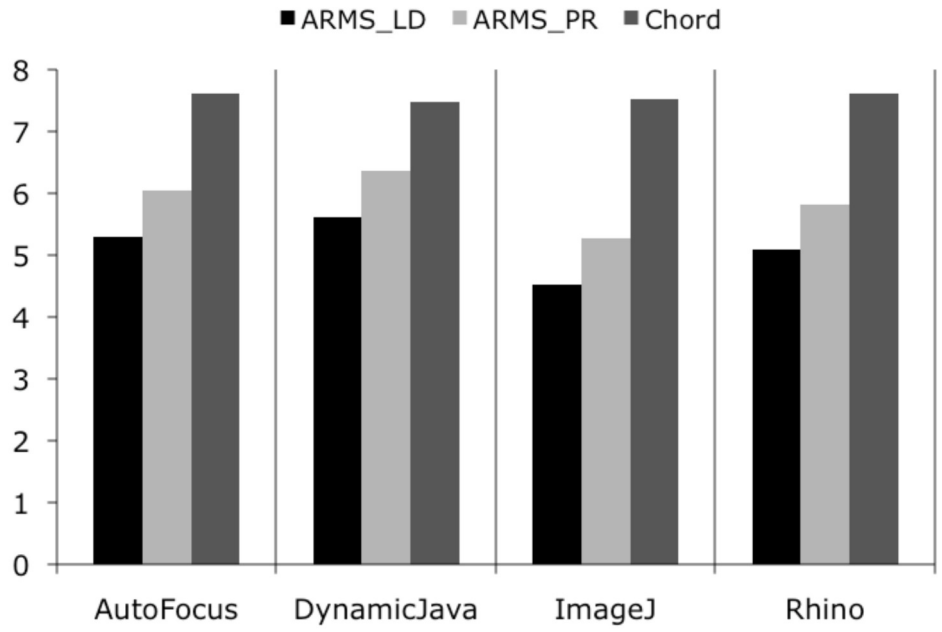

Figure 46: An illustration of the value of the mean lookup length (MLL) for the ARM_LD, the ARM_PR and the Chord simulated with four benchmark programs.

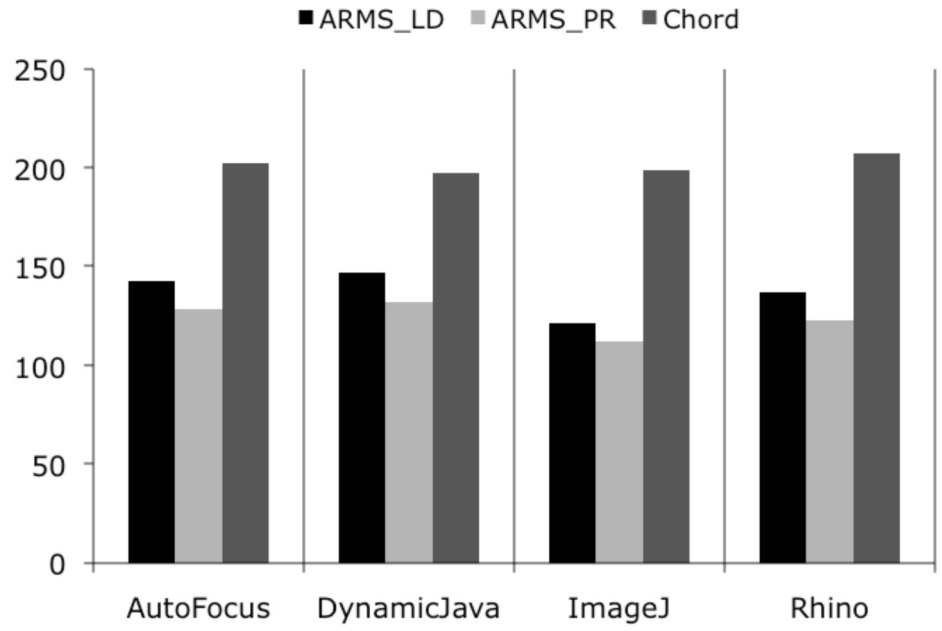

Figure 47: An illustration of the value of the distance per query (DPQ) for the ARM_LD, the ARM_PR and the Chord simulated with four benchmark programs.

Parameter	Benchmark	Network Size				
		1156	***1600***	***2116***	***2704***	***3364***
Execution time (ET)	AutoFocus	11055.2	12721.1	12626.3	14796.5	16523.8
	DynamicJava	42400.0	45720.9	53199.0	68678.9	69640.6
	ImageJ	12142.7	13563.0	14755.1	16677.3	17347.6
	Rhino	13096.2	14217.1	15052.6	16917.6	17722.3
Distance per query (DPQ)	AutoFocus	115.4	142.9	169.8	199.9	228.2
	DynamicJava	122.6	146.8	181.2	206.2	239.1
	ImageJ	106.6	121.8	142.9	160.9	192.6
	Rhino	111.4	137.2	167.6	194.7	224.0
Mean lookup length (MLL)	AutoFocus	5.025	5.306	5.536	5.678	5.851
	DynamicJava	5.374	5.613	5.892	5.908	6.210
	ImageJ	4.430	4.527	4.572	4.612	4.941
	Rhino	4.799	5.090	5.364	5.521	5.722
Avg. workload (AWL)	AutoFocus	45.31	36.97	30.70	25.52	21.79
	DynamicJava	8.29	6.89	5.80	4.92	4.40
	ImageJ	18.53	14.82	12.96	9.97	9.08
	Rhino	59.38	48.68	41.25	34.75	28.80
Workload standard deviation (WSD)	AutoFocus	26.87	23.75	21.57	18.90	17.22
	DynamicJava	8.54	7.96	7.08	6.35	5.74
	ImageJ	36.50	30.62	33.60	26.98	25.16
	Rhino	40.77	37.31	33.18	30.56	26.83

Table 9: Listing of the test results from simulating ARMS with the heuristic information designed for the lookup length under various network sizes.

Parameter	Benchmark	Network Size				
		1156	***1600***	***2116***	***2704***	***3364***
Execution time (ET)	AutoFocus	10287.9	12508.3	12580.2	15737.5	16368.7
	DynamicJava	43790.2	45816.6	56710.0	71827.2	66105.8
	ImageJ	13022.1	12502.8	15558.5	14226.0	16888.3
	Rhino	12575.4	13393.8	15583.4	16729.6	19663.6
Distance per query (DPQ)	AutoFocus	103.7	128.2	150.5	173.5	196.3
	DynamicJava	111.3	131.8	155.1	178.5	202.4
	ImageJ	90.4	112.3	126.7	142.2	166.6
	Rhino	99.7	123.2	147.1	169.8	191.7
Mean lookup length (MLL)	AutoFocus	5.749	6.055	6.319	6.513	6.647
	DynamicJava	6.158	6.367	6.578	6.681	6.930
	ImageJ	5.005	5.269	5.164	5.453	5.762
	Rhino	5.510	5.809	6.145	6.312	6.483
Avg. workload (AWL)	AutoFocus	52.20	42.10	34.74	29.11	24.57
	DynamicJava	9.36	7.67	6.36	5.42	4.79
	ImageJ	21.25	16.97	14.46	11.73	10.34
	Rhino	66.81	54.88	46.50	39.61	32.59
Workload standard deviation (WSD)	AutoFocus	34.21	31.44	28.77	26.86	25.66
	DynamicJava	10.16	10.00	8.79	7.77	7.33
	ImageJ	38.69	34.76	36.20	31.48	29.34
	Rhino	48.80	45.05	40.89	40.47	35.06

Table 10: Listing of the results from simulating ARMS with the heuristic information designed for the proximity under various network sizes.

6.2.2. Load balancing optimisation

As shown in the profiling study, the amount of messages sent or received by an object was not identical and thus hotspots can be developed in a structured overlay. Two kinds of hotspots have been defined in [42], namely *query hotspot* and *routing hotspot*. The query hotspots refer to popular objects frequently referenced by other objects. On the other hand, the routing hotspots refer to objects that send query messages much more than others do.

ARMS reduces the number of hotspots in the system through two approaches. Firstly, a query message can be transported via multiple routes taking the advantage of the ARMS flexible structure. Secondly, heuristic information that directs query messages towards lowly utilised nodes can be used in order to evenly distribute message workloads.

The heuristic information is computed by replacing the Manhattan distance H_{ij} with the runtime workloads for each node, ω_{ij},

$$H_{ij} = \frac{1}{\omega_{ij}}$$

Table 11 summarises the values of the performance measurements for the ARM_LB. Figure 48 compares the values of WSD in the ARM_LD, the ARM_LB and the Chord. The values of WSD declined as the increase of the network size. The observation is expected because there were more nodes available for query processing in the system. The figure shows that the standard deviation of the number of queries processed at each

node was lower in the ARM_LB than that of the ARM_LD. This implies that the ARM_LB caused query messages distributed more evenly than the ARM_LD did. Amongst the three schemes, the Chord had the lowest quantity since it requires no parallel query message as opposite to the two ARMS methods.

However, the ARM_LB produced the higher values of MLL and AWL as illustrated in Figure 49 and Figure 50. This is because a query message may need to route with a longer path in order to allocate workloads uniformly across neighbouring nodes.

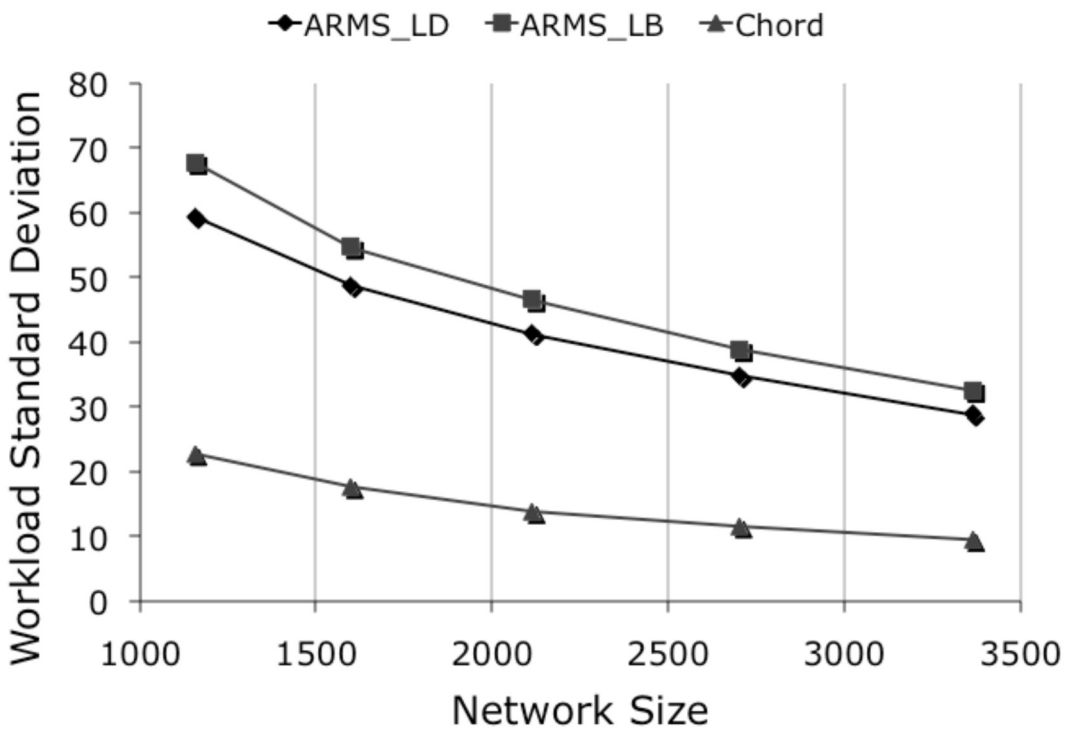

Figure 48: An illustration of the value of the workload standard deviation (WSD) for the ARM_LD, the ARM_LB and the Chord through simulating the benchmark Rhino.

Parameter	Benchmark	Network Size				
		1156	***1600***	***2116***	***2704***	***3364***
Execution time (ET)	AutoFocus	11971.1	13942.1	15209.1	16522.7	19004.0
	DynamicJava	46526.0	46807.7	62571.3	73809.6	76337.3
	ImageJ	13180.3	13876.8	15499.3	15144.0	17853.5
	Rhino	14794.3	16634.7	17860.4	20656.6	21203.4
Distance per query (DPQ)	AutoFocus	131.6	164.1	195.3	229.8	260.6
	DynamicJava	140.2	169.1	210.8	237.0	270.0
	ImageJ	117.2	139.5	166.3	189.6	228.3
	Rhino	127.0	158.0	193.4	221.8	255.2
Mean lookup length (MLL)	AutoFocus	5.706	6.028	6.253	6.466	6.641
	DynamicJava	6.237	6.425	6.743	6.765	6.963
	ImageJ	4.959	5.106	5.125	5.318	5.668
	Rhino	5.480	5.760	6.101	6.233	6.450
Avg. workload (AWL)	AutoFocus	51.83	41.89	34.39	29.01	24.71
	DynamicJava	9.46	7.74	6.49	5.46	4.82
	ImageJ	21.24	16.38	14.07	10.83	10.28
	Rhino	67.75	54.66	46.51	38.83	32.56
Workload standard deviation (WSD)	AutoFocus	21.31	20.99	18.77	17.13	16.33
	DynamicJava	7.18	7.31	6.63	5.75	5.34
	ImageJ	33.08	26.50	29.17	22.23	23.49
	Rhino	34.08	30.87	27.53	26.47	23.63

Table 11: A demonstration of the test results for simulating ARMS with the heuristic information designed for load balancing under various network sizes.

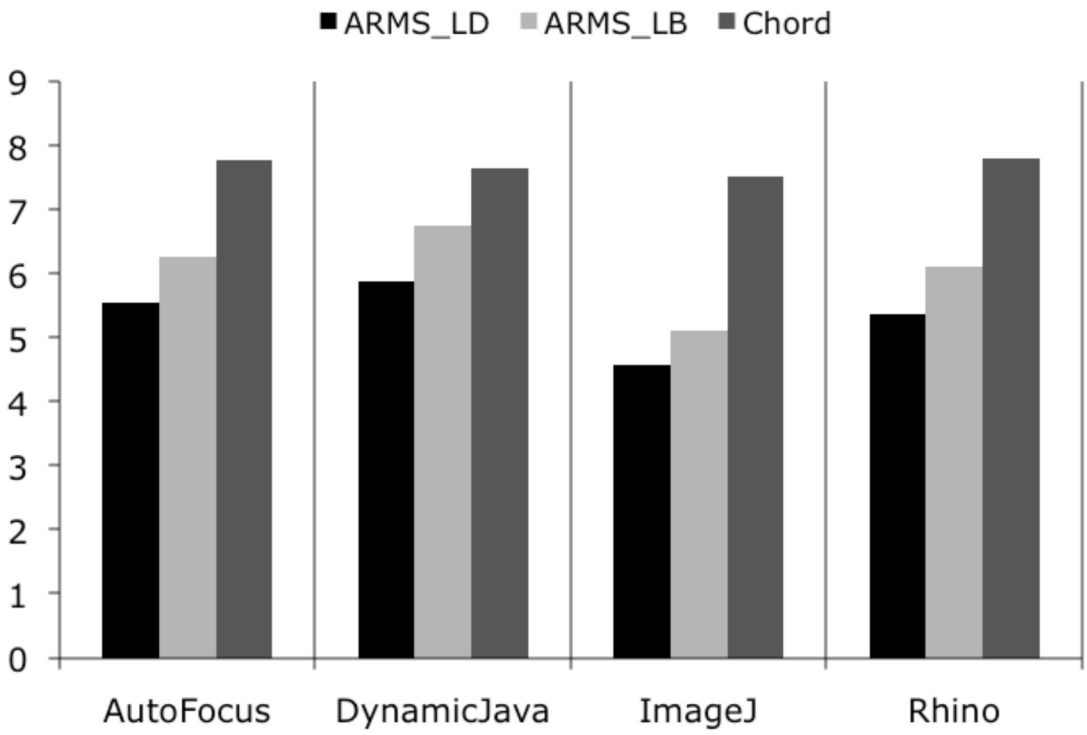

Figure 49: An illustration of the values of the mean lookup length (MLL) for the ARM_LD, the ARM_LB and the Chord simulated with four benchmark programs.

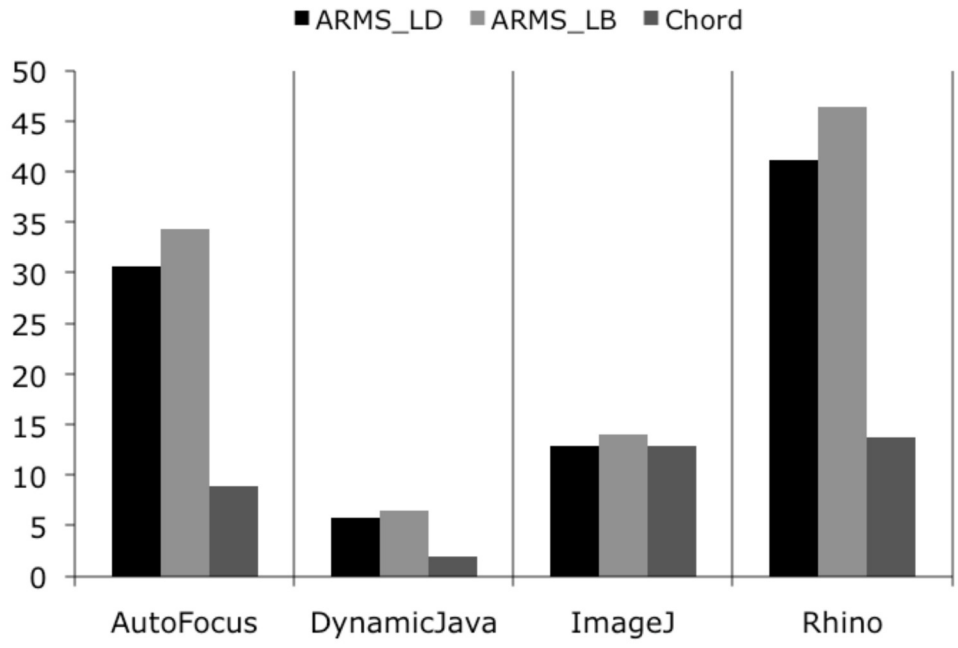

Figure 50: An illustration of the value of the average workload (AWL) for the ARM_LD, the ARM_LB and the Chord simulated with four benchmark programs.

6.2.3. Degree of connectivity

Intuitively, the objects communicated more frequently should have more knowledge of the addresses of other objects. Consequently, the nodes that contain these popular objects are potential candidates for processing queries although the workloads of these nodes may be higher than the rest. Here, the usefulness of the heuristic information is investigated based on the degree of connectivity for the lookup efficiency, which is given by

$$\eta_{ij} = \frac{\Omega_{ij}}{\sum_{C_{ik} \in N(S^P)} \Omega_{ik}}, \text{ if } C_{ij} \in N(s^p)$$

where Ω_{ij} is the value of the degree of connectivity for each node.

Table 12 illustrates the values of the performance measurements for the ARM_DOC. First of all, the tests showed for the execution time differently in various applications as illustrated in Figure 51. The ratio of the execution time of the ARM_DOC to the execution time of the ARM_LD was 119.5%, 112%, 95.3% and 105% for the AutoFocus, the DynamicJava, the ImageJ and the Rhino respectively. Accordingly, as demonstrated in the profiling results, the standard deviation of the number of outgoing calls was 11.9 (AutoFocus), 10.4 (DynamicJava), 151.2 (ImageJ) and 22.8 (Rhino). In other words, the ARM_DOC gave the best performance for the benchmark that had the most outspread object invocations.

Figure 52 and Figure 53 show the quantities of AWL and WSD for the ARM_DOC and the ARM_LD. The value of AWL was higher in the ARM_DOC than the ARM_LD as

the nodes needed to process additional queries due to the longer lookup length. The value of WSD was also larger in the ARM_DOC as expected.

Parameter	Benchmark	Network Size				
		1156	***1600***	***2116***	***2704***	***3364***
Execution time (ET)	AutoFocus	10873.8	12364.3	14840.8	17680.2	18292.3
	DynamicJava	46287.3	51027.8	59832.2	76944.4	80347.4
	ImageJ	13446.5	15180.7	15057.6	15895.9	17751.5
	Rhino	13775.3	16000.0	16400.4	17759.0	23254.1
Distance per query (DPQ)	AutoFocus	128.8	160.5	190.3	223.0	252.1
	DynamicJava	135.0	165.2	191.4	225.0	267.3
	ImageJ	110.4	133.4	154.6	180.6	209.8
	Rhino	125.4	155.4	185.9	216.3	246.8
Mean lookup length (MLL)	AutoFocus	5.656	5.942	6.158	6.367	6.501
	DynamicJava	5.978	6.354	6.370	6.506	6.870
	ImageJ	4.661	4.762	4.953	5.181	5.369
	Rhino	5.450	5.631	5.961	6.109	6.244
Avg. workload (AWL)	AutoFocus	51.45	41.34	33.80	28.58	23.93
	DynamicJava	9.13	7.66	6.20	5.30	4.76
	ImageJ	19.54	15.62	13.57	10.39	9.96
	Rhino	67.41	53.02	45.56	38.05	31.44
Workload standard deviation (WSD)	AutoFocus	42.16	38.78	34.10	31.37	29.70
	DynamicJava	10.72	10.94	9.46	8.35	7.56
	ImageJ	41.50	38.05	34.66	28.33	29.85
	Rhino	60.58	53.17	47.60	46.16	42.02

Table 12: Listing of test results for simulating ARMS with the heuristic information designed for the degree of connectivity under various network sizes.

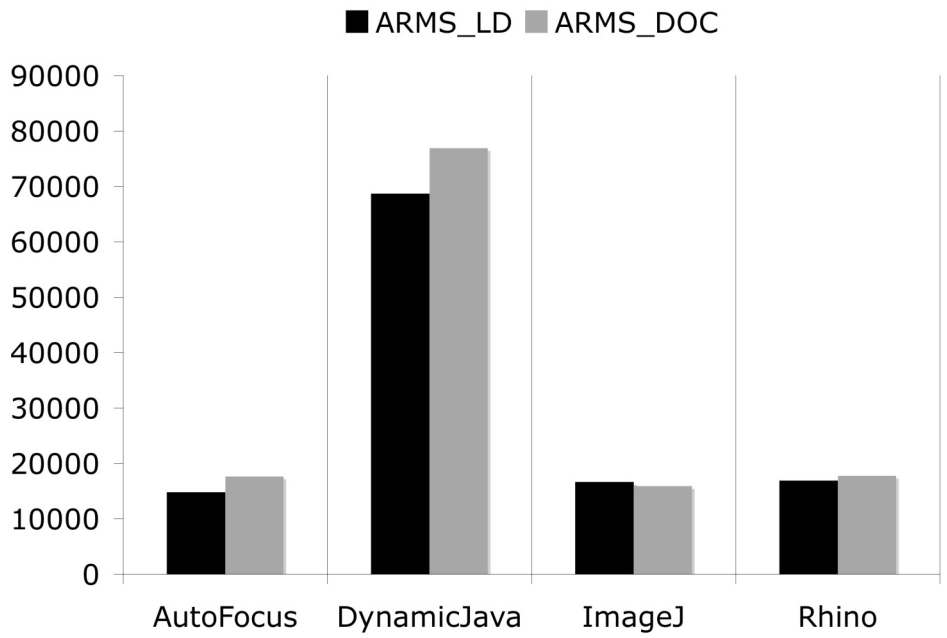

Figure 51: An illustration of the execution time for all benchmark programs with simulating the ARM_LD and the ARM_DOC on the network of 2704 nodes.

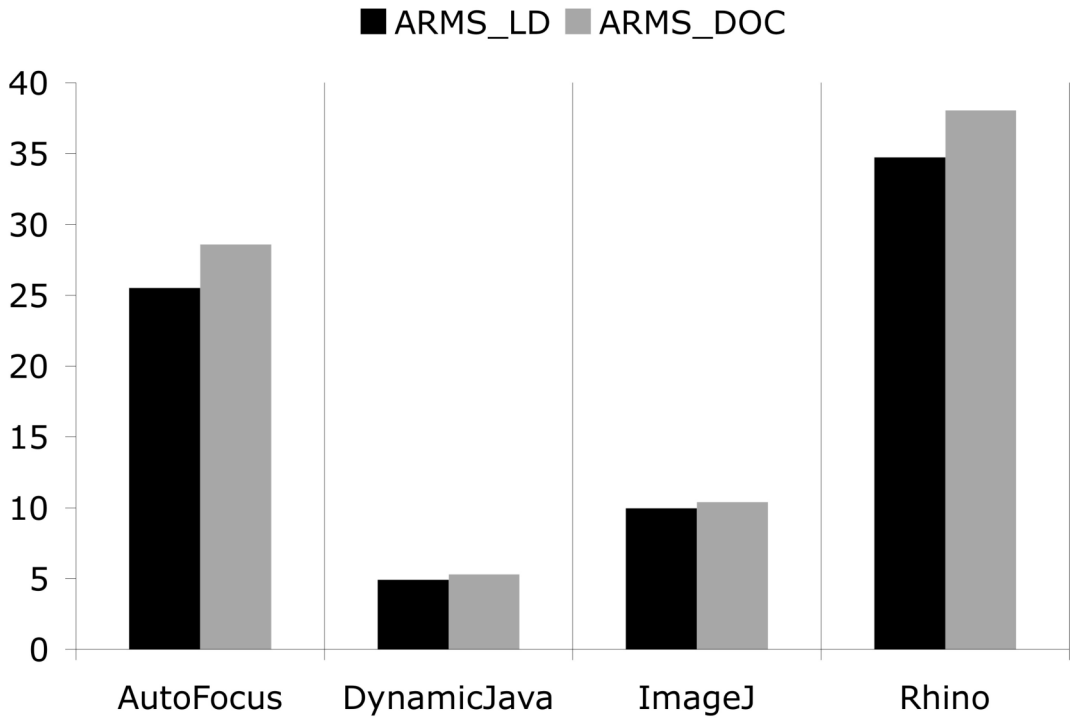

Figure 52: An illustration of the average workload for all benchmark programs with simulating the ARM_LD and the ARM_DOC on the network of 2704 nodes.

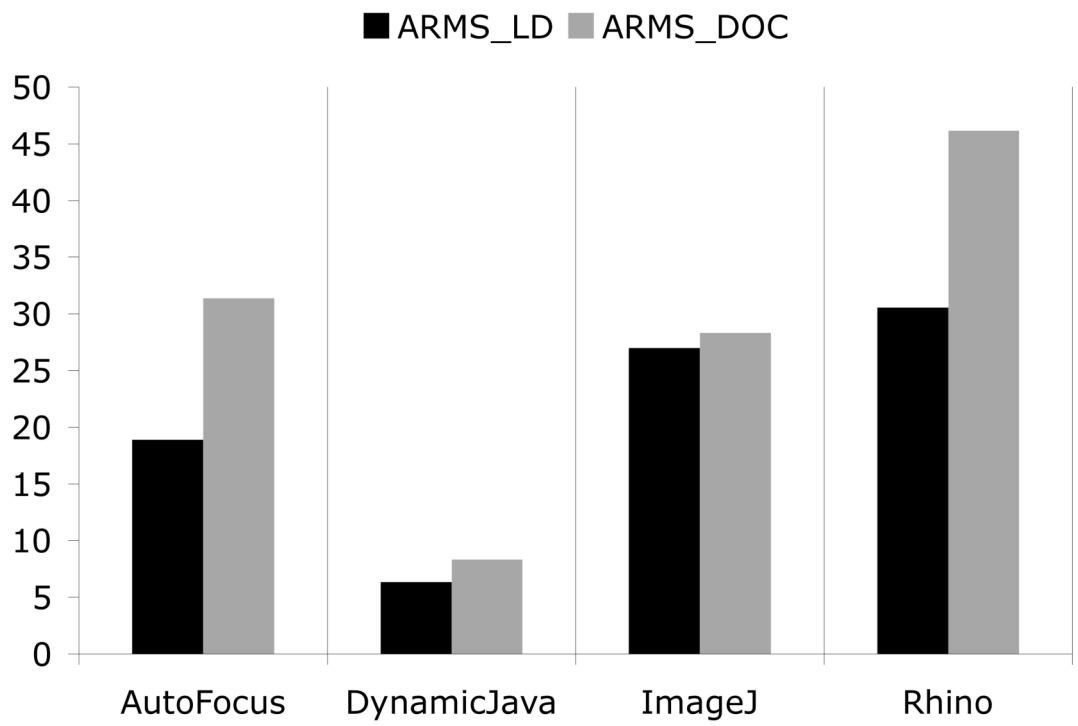

Figure 53: An illustration of the standard deviation for workloads for all benchmark programs with simulating the ARM_LD and the ARM_DOC on the network of 2704 nodes.

6.2.4. Optimisation based on spatial locality

As illustrated in Table 13, an object is more likely communicated to its class objects or inherited objects. Therefore, an optimisation strategy was developed by utilising the spatial relationship between objects. With this strategy, the pheromones between associated objects will be enhanced and thus this makes communications exchanged between those associated objects much faster. In other words, the idea of the spatial strategy is to build clusters of objects in the network and subsequently to strengthen the links within each cluster.

In Java, the identifier of an object consists of two components: instance key and class name. The identifiers of two objects have the same class name if they are created from the same class; further, two objects have the same instance key if they are belonged to the inheritance tree. Consequently, the object identifier can be used to determine the relationship of two objects. The heuristic value H_{ij} can be modelled by a step function:

if two objects are related, H_{ij} is assigned to a constant; otherwise, it is assigned to a zero.

$$H_{ij} = \begin{cases} C_H & \text{if two objects are related} \\ 0 & \text{otherwise} \end{cases}$$

The test results are presented in Table 14. It is noted that the mean lookup length was influenced by the pattern of message passing. Figure 54 compares the MLL values of the ARM_SL with the MLL values of the ARM_LD. In particular, the ratio of the ARM_SL lookup lengths to the ARM_LS lookup lengths was 1.146, 1.128, 1.109 and 1.134 respectively for the AutoFocus, the DynamicJava, the ImageJ and the Rhino. That is, the ARM_SL scheme was able to locate the ImageJ objects faster than the others did because the benchmark ImageJ had the highest number of the spatial calls.

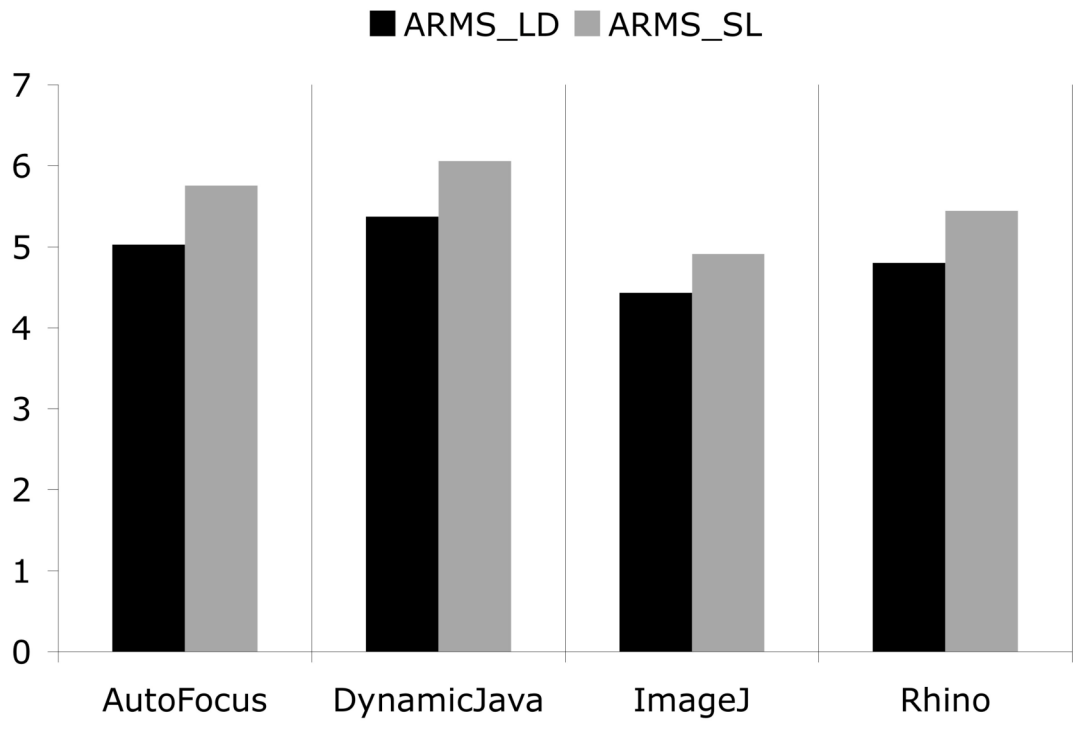

Figure 54: An illustration of the mean lookup length for the ARM_LD and the ARM_SL simulated with four benchmark programs on the network of 1156 nodes.

Parameter	Benchmark Program			
	AutoFocus	***DynamicJava***	***ImageJ***	***Rhino***
Total Remote Calls	2623	531	4636	4417
Spatial calls	84 (3.20%)	237 (44.63%)	2087 (45.02%)	786 (17.79%)
Class calls	0 (0%)	9 (1.69%)	1751 (37.77%)	21 (0.48%)
Inheritance calls	84 (3.20%)	228 (42.94%)	336 (7.25%)	765 (17.32%)

Table 13: Listing of the profiling results for four benchmark programs regarding the total remote calls, the spatial calls, the class calls and the inheritance calls.

Parameter	Benchmark	Network Size				
		1156	***1600***	***2116***	***2704***	***3364***
Execution time (ET)	AutoFocus	10719.2	11610.6	13666.7	14609.7	17015.1
	DynamicJava	42273.3	46729.8	57228.0	69747.1	72609.4
	ImageJ	12620.7	12690.4	14746.3	15483.6	16062.9
	Rhino	13770.0	15370.7	15393.9	17809.5	19499.2
Distance per query (DPQ)	AutoFocus	132.1	164.9	193.3	227.4	256.7
	DynamicJava	137.3	168.4	204.8	234.1	269.4
	ImageJ	116.6	137.0	160.4	190.9	207.0
	Rhino	125.4	156.8	190.8	218.8	250.5
Mean lookup length (MLL)	AutoFocus	5.757	6.074	6.242	6.450	6.598
	DynamicJava	6.061	6.366	6.641	6.725	6.946
	ImageJ	4.912	5.050	5.055	5.432	5.371
	Rhino	5.444	5.754	6.082	6.196	6.395
Avg. workload (AWL)	AutoFocus	52.22	42.09	34.24	28.78	24.38
	DynamicJava	9.20	7.66	6.38	5.43	4.80
	ImageJ	19.38	16.50	13.34	10.63	9.75
	Rhino	67.11	54.41	46.12	38.80	31.78
Workload standard deviation (WSD)	AutoFocus	24.18	22.31	20.10	17.83	16.35
	DynamicJava	7.41	7.52	6.62	5.90	5.51
	ImageJ	29.41	28.50	26.46	20.81	22.59
	Rhino	41.83	36.83	34.30	32.96	28.68

Table 14: Listing of the test results for simulating ARMS with the heuristic information designed for the spatial locality in various network sizes.

6.3. Performance Analysis

Here, the performance of different heuristic information is compared. A new scheme is also introduced, termed ARM_AVG, where it simply takes the average value of the five forms of the heuristic information, namely the lookup distance, the proximity of nodes, the degree of connectivity, the spatial locality and the load balancing. The motivation is to study the potential of the use of a hybrid scheme in ARMS. Again, the Chord had been used for benchmarking in the simulation tests. The value of the mean lookup length for each scheme is illustrated in Figure 55 – 58 as the function of the network size. Obviously, all schemes based on ARMS were superior to the Chord due to the small radius of the randomised network and the efficient locating protocol.

The tests further showed that the heuristic based on the degree of connectivity resulted in a relatively short lookup path on the virtual structure. Using the heuristic rule increased the likelihood of passing queries through the nodes including objects that had the most connections in the system. Such nodes act as *super nodes*, which create shortcuts for the query forwarding.

Figure 59 – 62 show the quantities of the execution time of simulating the benchmark programs under each scheme. First of all, the execution time ascended as the increasing network size. However, the lookup path of the Chord was significantly longer than ARMS while simulating four benchmark programs, regardless the heuristic rule used in ARMS.

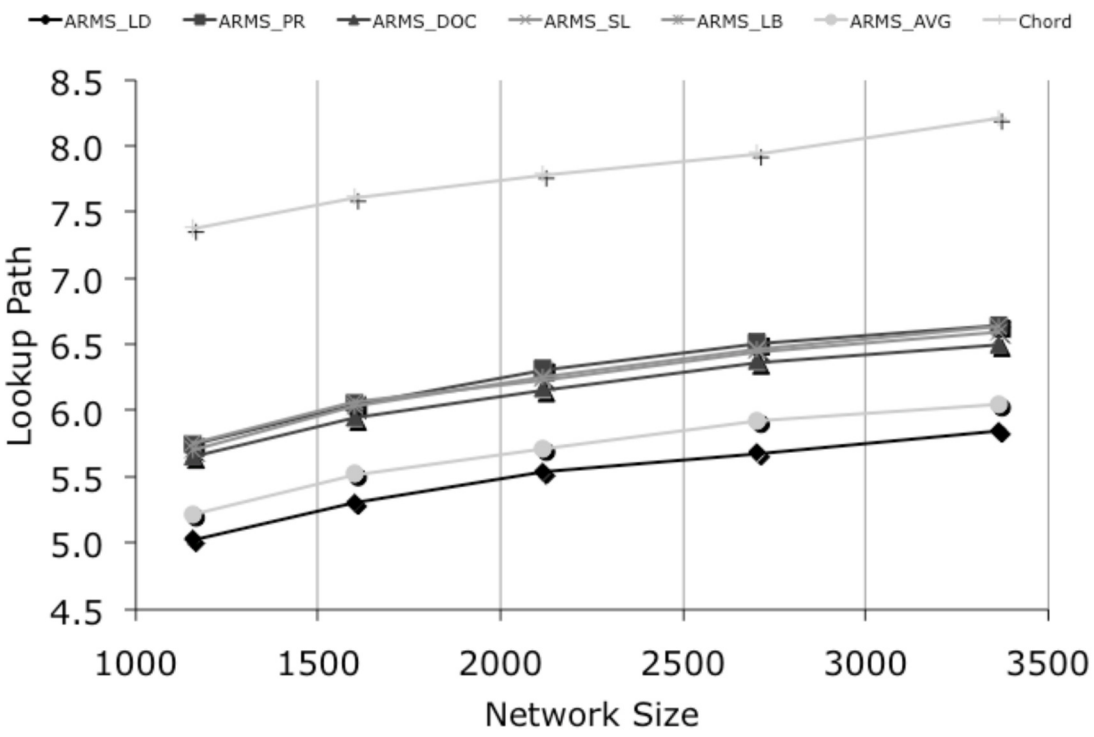

Figure 55: An illustration of the mean lookup length for all schemes as the function of the network size though simulating the benchmark AutoFocus.

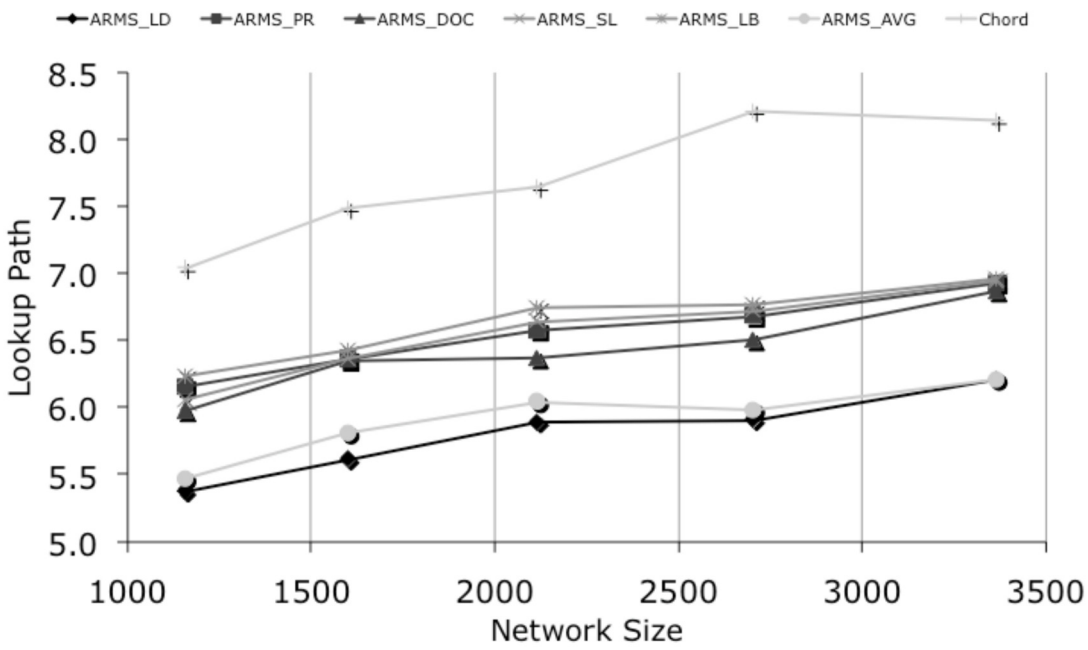

Figure 56: An illustration of the mean lookup length for all schemes as the function of the network size though simulating the benchmark DynamicJava.

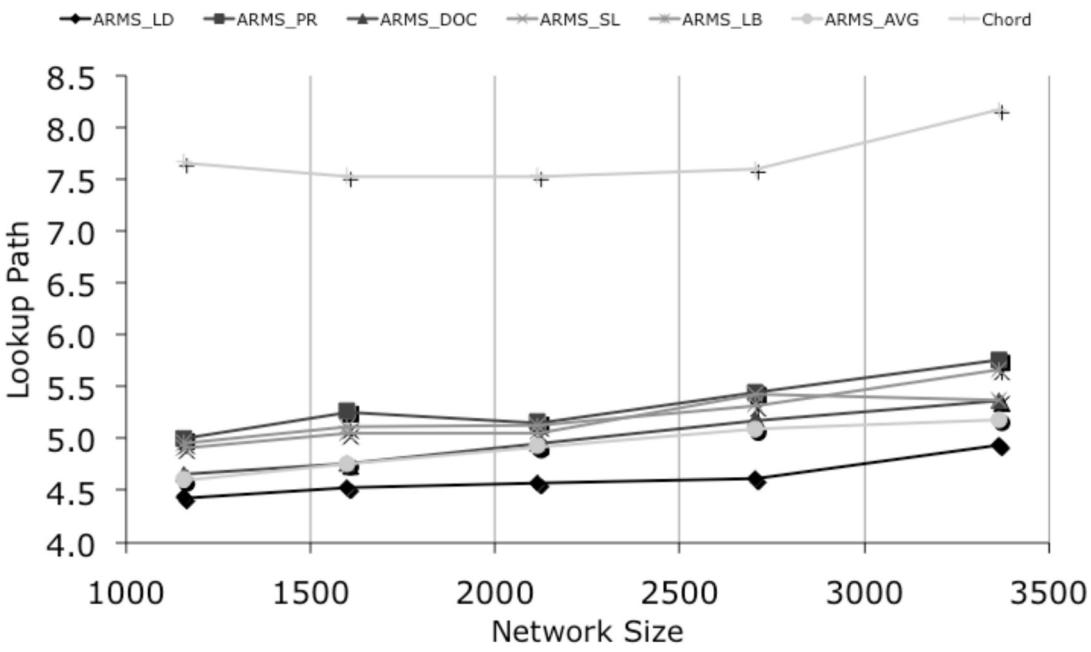

Figure 57: An illustration of the mean lookup length for all schemes as the function of the network size though simulating the benchmark ImageJ.

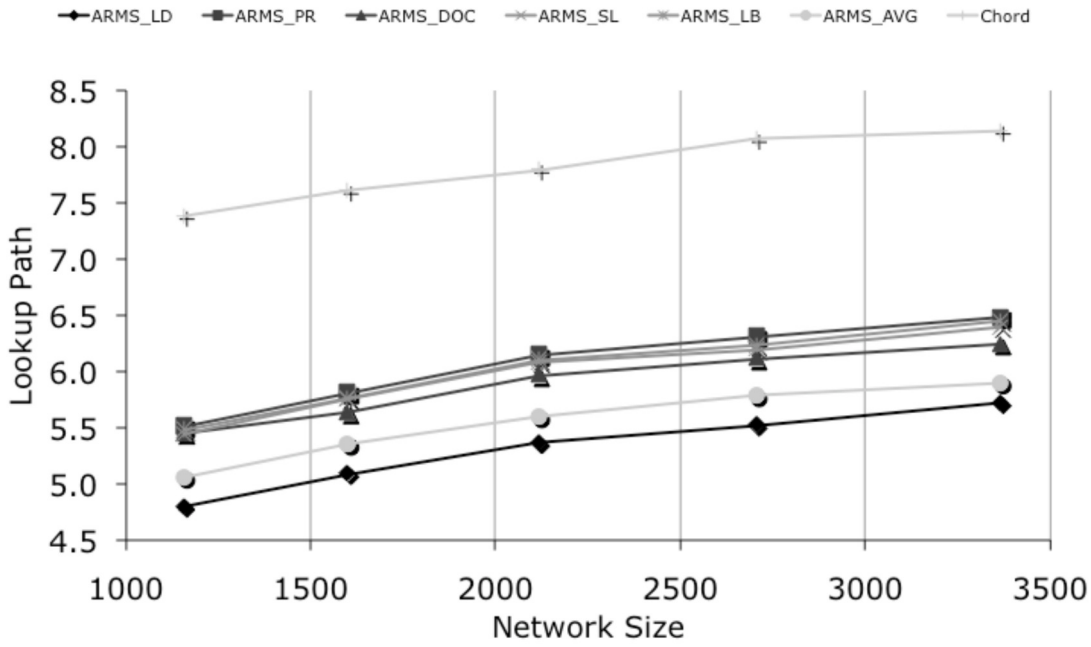

Figure 58: An illustration of the mean lookup length for all schemes as the function of the network size though simulating the benchmark Rhino.

The ARM_LD and the ARM_PR showed the competitive values for the execution time as the average improvement in execution time for the ARM_LD and the ARM_PR was 21.2% and 21.6% over the Chord. However, they approached the issue of the execution efficiency differently. While it led to a long travelling distance, the ARM_LD produced the shortest lookup path in the virtual organisation as illustrated in Figure 63. On the contrary, the ARM_PR emphasises on the minimum travelling distance on the physical distance. Hence, the value of distance per query (DPQ) was the lowest with the ARM_PR. As illustrated in Figure 64, the forwarding distance on the ARMS randomised overlay was averagely 61% as much as the forwarding distance on the Chord structured overlay. However, the efficiency of the ARM_PR compensates with longer lookup paths on the randomised overlay.

Next, the ARM_SL and the ARM_DOC are designed based on the characteristics of the object communication. The ARM_SL exploits the spatial property of message passing by strengthening the pheromones of the links between the spatial objects. The average completion time of the ARM_SL was 20.4% over the Chord, where it was the best result after the ARM_PR and the ARM_LD.

The ARM_DOC, on the other hand, takes advantage of objects with the high degree of connectivity. Those objects effectively shortened the overall distance between objects in the overlay network. Thus, the execution time of the ARM_DOC was 14.2% quicker than the execution time of the Chord. The weakness of the scheme, however, was its unbalanced distribution of message workloads as shown in Figure 65.

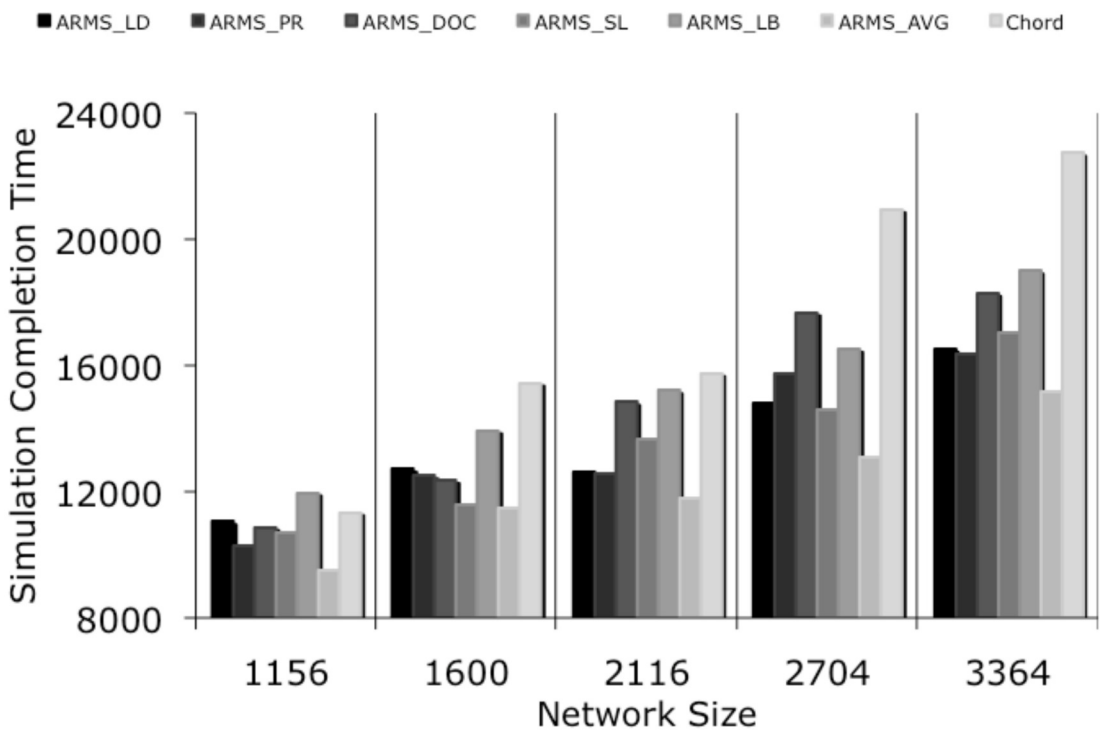

Figure 59: An illustration of the execution time for all schemes as the function of the network size based on the benchmark AutoFocus.

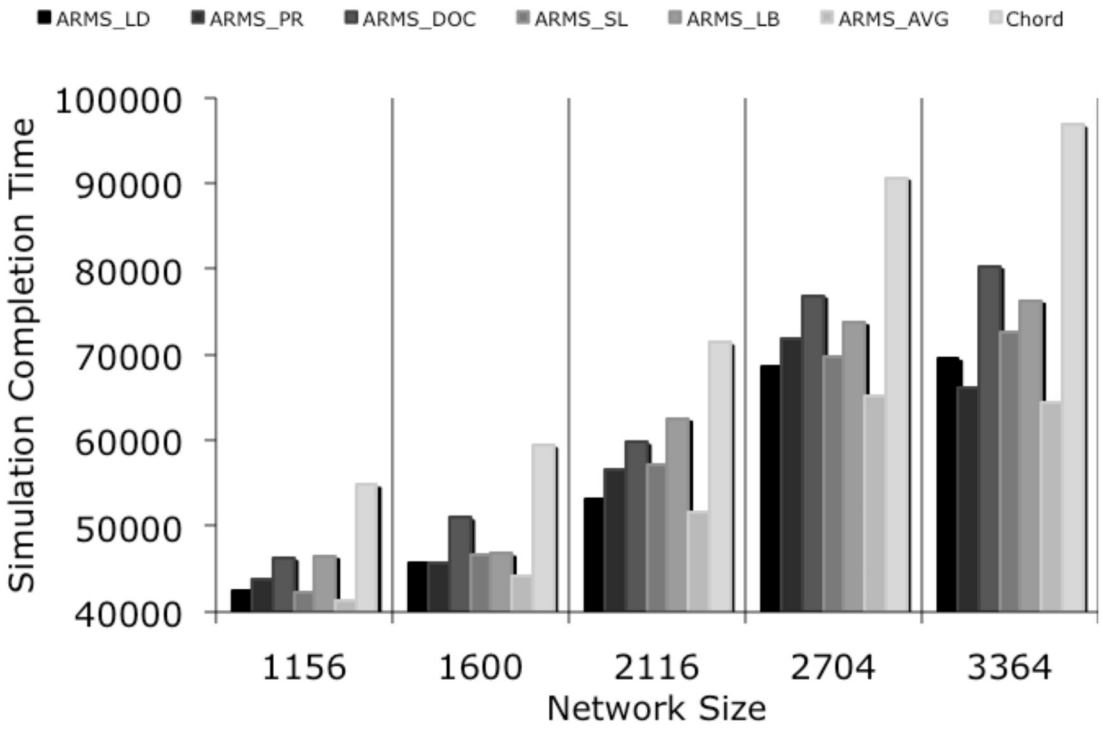

Figure 60: An illustration of the execution time for all schemes as the function of the network size based on the benchmark DynamicJava.

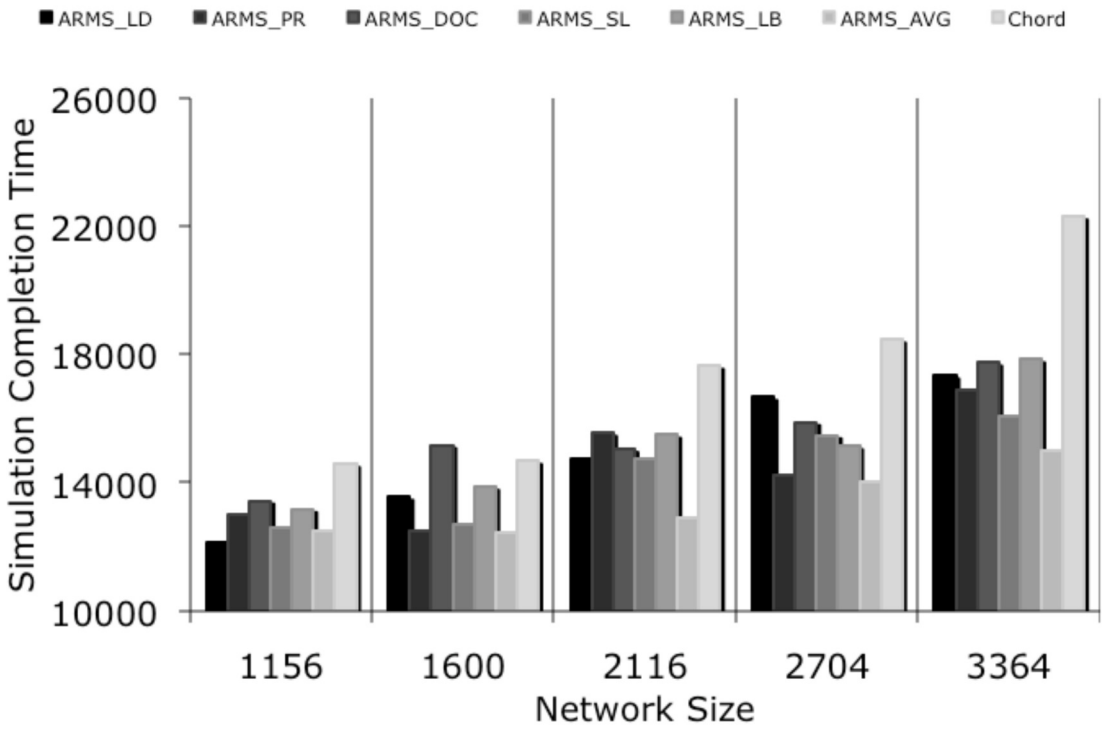

Figure 61: An illustration of the execution time for all schemes as the function of the network size based on the benchmark ImageJ.

The ARM_LB is designed for balancing workloads across nodes in the system. As depicted in Figure 65, the standard deviation of the workloads for the ARM_LB was significantly lower than the other four schemes. However, the cost of even workloads was the longer lookup path and travelling distance as depicted in Figure 63 and 64. Consequently, the execution time of the ARM_LB was the highest amongst five schemes, while it was still 13.2% faster than the Chord.

The mixed scheme, ARM_AVG, had also been investigated. The ARM_AVG simply takes the average of the five heuristic values. The tests showed that the ARM_AVG was superior to other schemes regarding all performance measurements, as depicted in Figure 63 – 65. As the result, the execution time of the ARM_AVG was on average 26.8% faster than the execution time of the Chord. Because the simple mixed scheme showed a significant improvement in the efficiency of the object locating, study of a

sophisticated approach for adapting the dynamic message patterns and the changing network conditions is desired.

For instance, one can parameterise the heuristic information. Then, a decision tree could be used to control the importance of different kinds of the heuristics. Through this, the efficiency of the algorithm can be tuned based on the characteristics of the object communication. A profiler is therefore useful in the collection of runtime information for a target program.

Alternatively, a dynamic approach could be used to adjust the importance of the heuristic rules at runtime. However, this may require minor changes in the updating algorithm of ARMS.

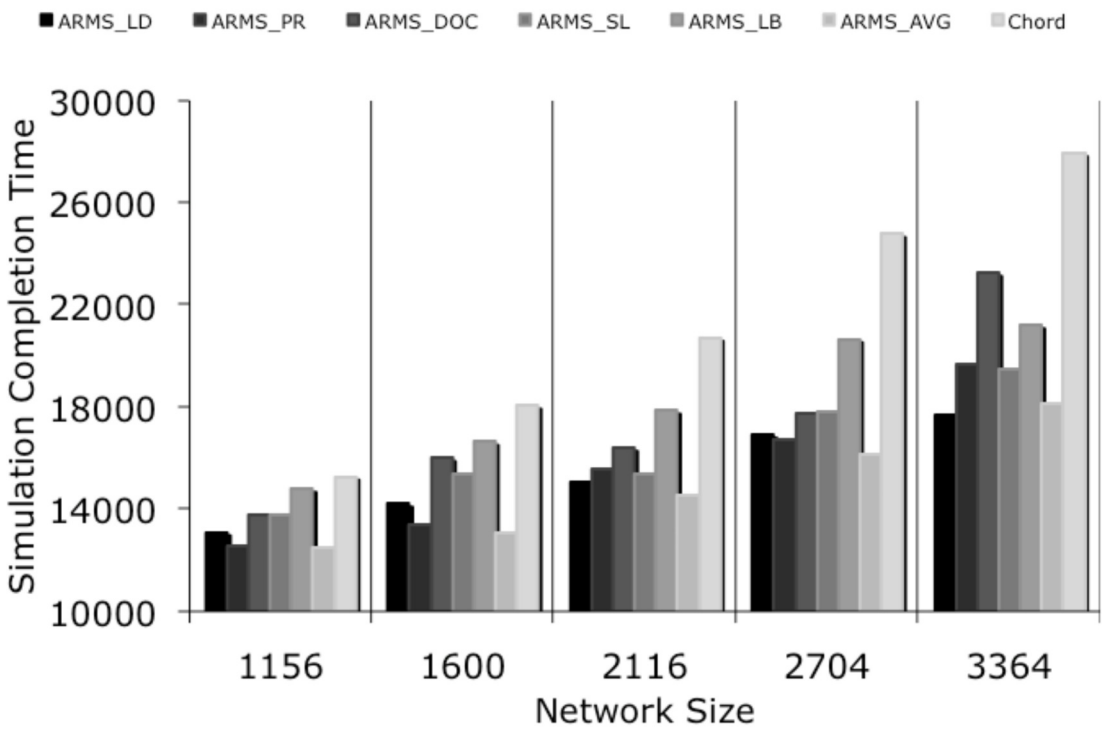

Figure 62: An illustration of the execution time for all schemes as the function of the network size based on the benchmark Rhino.

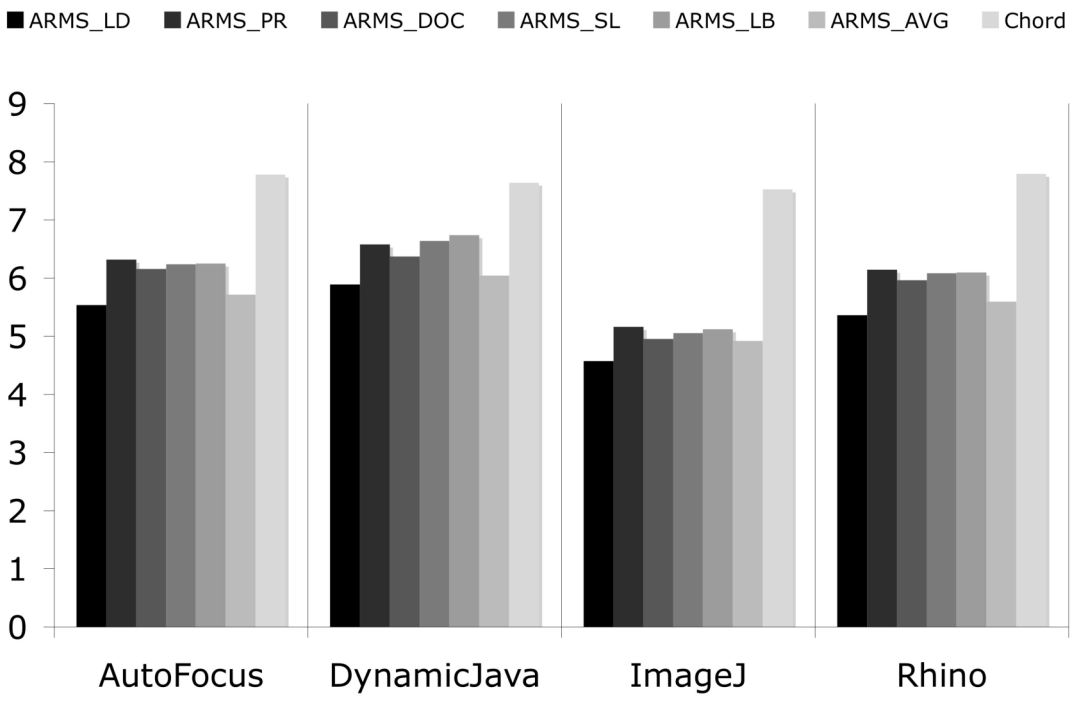

Figure 63: An illustration of the mean lookup length for all schemes as simulated on a network of 2116 nodes.

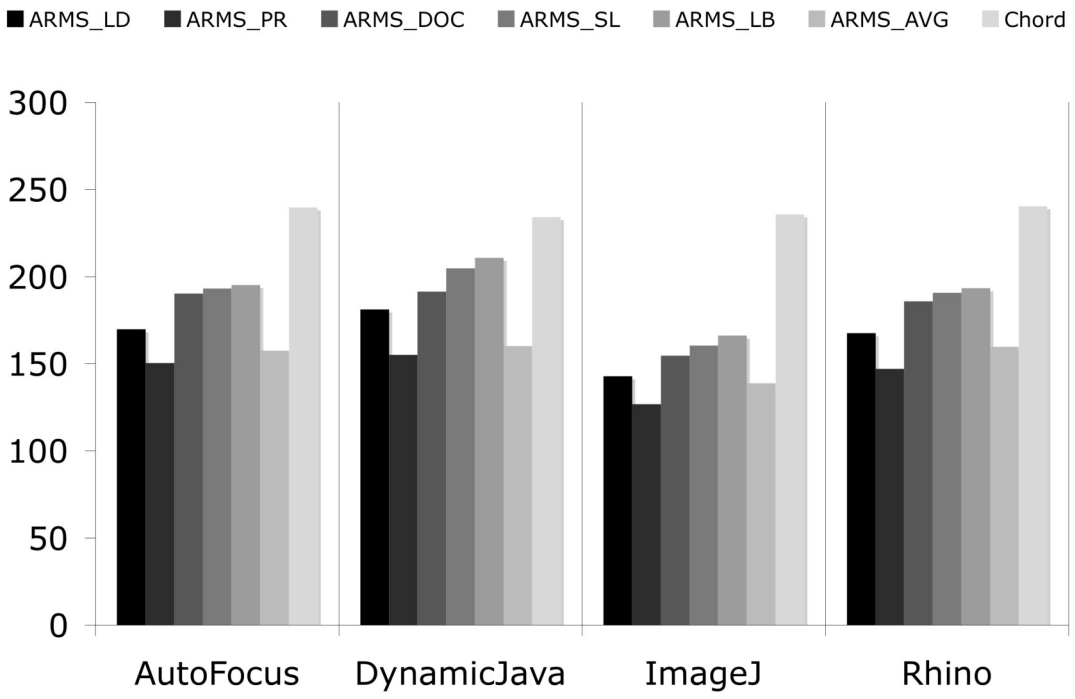

Figure 64: An illustration of the distance per query (DPQ) for all schemes as simulated on a network of 2116 nodes.

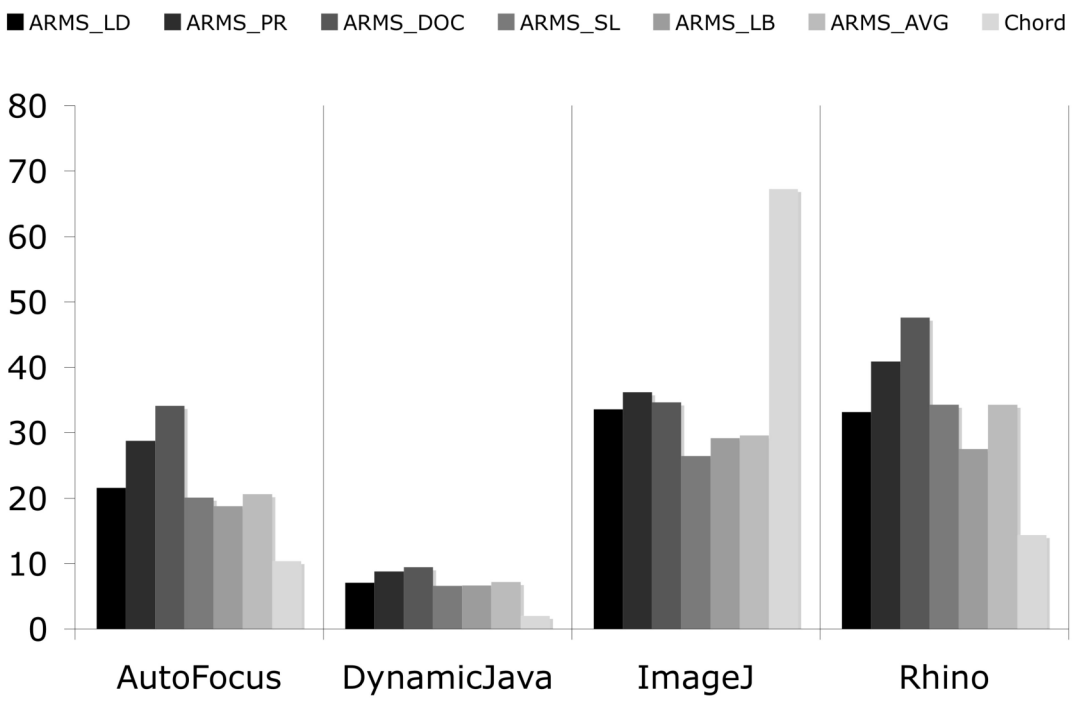

Figure 65: An illustration of the standard deviation for workloads (WSD) for all schemes as simulated on a network of 2116 nodes.

7. A Hybrid Scheme for Locating Migrated Objects

7.1. Overview of Relocation Schemes

ARMS consists of two important protocols: the *locating protocol* and the *relocation protocol*. Earlier chapters have presented a simulation study of the characteristics of the locating protocol. In this chapter, the features of the relocation protocol are examined. In general, resource management enables dynamic load distribution, fault resilience, flexible system administration, and data access locality. Agha [44] observed two effective techniques for the resource management: *replication* and *migration*. Since replication techniques have been examined in other P2P overlay networks [23, 34, 38], migration techniques are the focus of this study. It is ideal to make a migration process transparent to the developer so that it can reduce the programming complexity. Consequently, it is critical to provide transparent accesses to migrated objects as if migration never occurred to ensure the continuance of communications. Such a process is termed *transparent object relocation*. ARMS defines a transparent relocation scheme that combines three common approaches: *home based*, *forward pointers* and *backtracking* [34, 49, 138, 139]. These approaches are demonstrated in Figure 66.

With the home-based approach, each object is assigned to a fixed server, which is usually the birthplace of that object. The name of the object contains the address of the birthplace and hence messages are transported to the object via the home server. On every migration, an object needs to synchronise with its birthplace. For instance, before an object is migrated from Node *1* to Node *2*, as illustrated in Figure 66-A, a notification is sent to the home of the migrated object to update the new address. Obviously, this can cause significant overheads in the migration process.

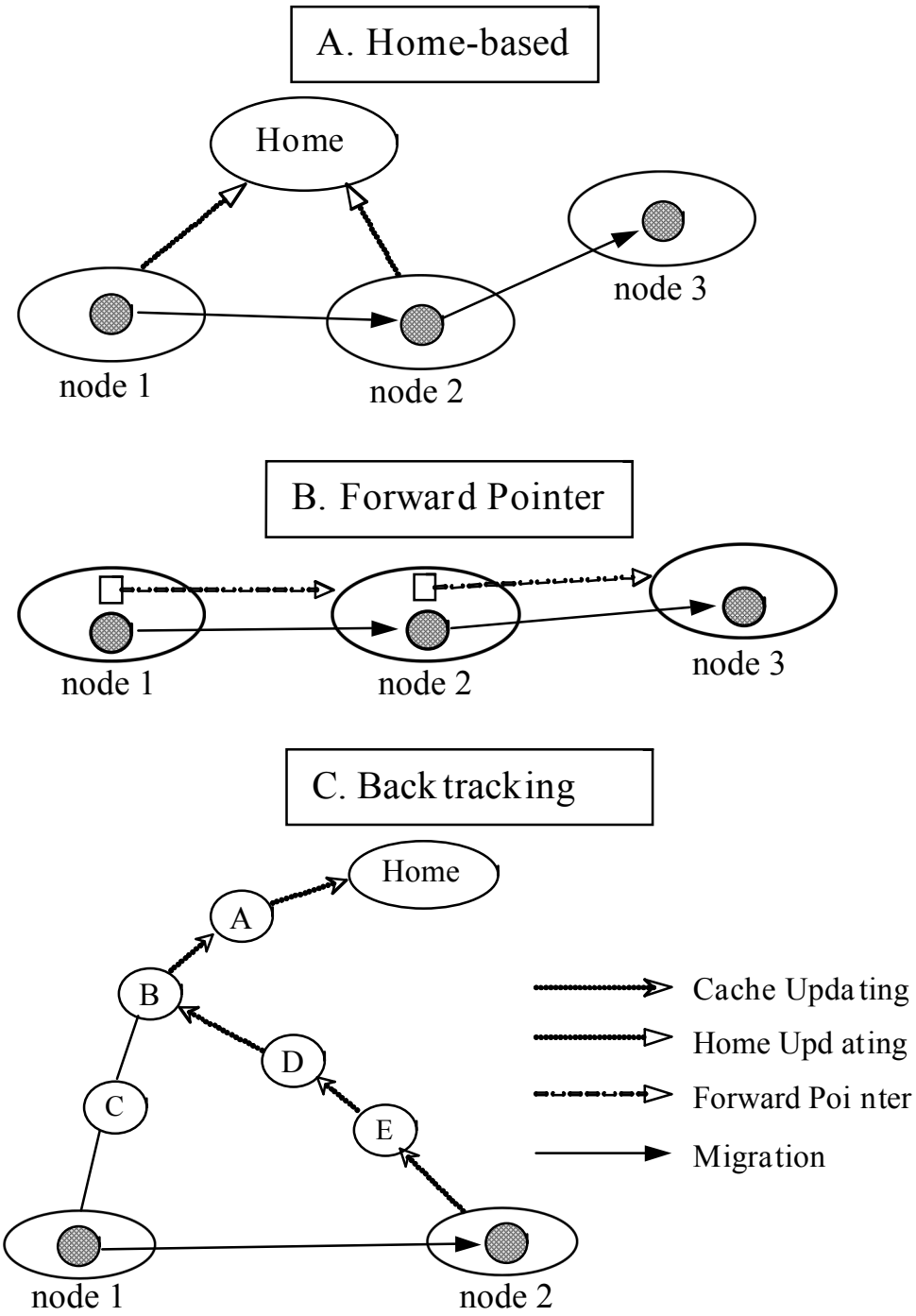

Figure 66: An illustration of the three common relocation schemes: (A) home-based; (B) forward pointers; (C) backtracking.

In addition, the distance of the home peer – the birthplace of the object – to the migration object has an effect on the efficiency of the home-based scheme. Therefore, a query may have to transverse a longer route to the new location when the migration object and the source object of the message are both separated by large distances from the home location, but close to each other on the physical network. This is commonly known as the trombone problem [140].

The backtracking can help to improve the trombone problem by caching the new location of the migrated object at peers along the path between the home peer and the migrated object. The backtracking is commonly used in structured networks to provide shortcuts to a nearby copy of the referenced data. Due to the consistency of the structured overlay topology, queries will eventually fall on the same path as they are approaching to the home peer. Subsequently, it could be more efficient to cache the address of the migrated object at those peers so that it can shorten the number of steps required in the lookup process. As shown in Figure 66-C, an update message is routed via a path toward the home peer, i.e. Peer *A*, *B*, *D* and *E*. Then any of these peers can forward subsequent queries to the new location on behalf of the home peer. Therefore, the approach not only shortens the search path but also reduces the workloads at the home peer.

However, the backtracking still incurs high migration overheads due to the migration synchronisation. Alternatively, a forwarding pointer can be used to redirect queries to the new location without the need of a home peer. With forwarding pointers, a migrated object leaves a forwarding address at the last node after migration. Succeeding queries will follow the chain of forwarding addresses until they reach the migrated object. Therefore, forwarding pointers can be used temporally to redirect messages to the new

location until the updating message issued by the peer to which the object has been relocated, reaches the home peer. This can lead to improvement of the migration process. Because the chain of pointers is growing with the number of object migrations, it is not practical for a large number of migrations as it will lead to an extremely long forwarding chain. Such a long chain is undesirable and risky. Firstly, the long forward chain can significantly degrade the relocation efficiency. Secondly, the long chain dramatically increases the complexity of the path maintenance as any broken segment can generate extra delays in query forwarding or even result in query lost. Finally, the long forward chain increases the possibility of *live locks* in the network. Live lock refers to a loop formed by a long forward chain, which it causes a query to be forwarded endlessly inside the loop.

In ARMS, the excessively long chain is resolved via the backtracking. An update message returned from the destination via the forward chain will force the peers to revise their forwarding pointers. As a result, these peers will contain the latest location of the migrated object. The updated forwarding pointers will subsequently become fingers of the forwarding table, which provide shortcut links to the migrated object. The redundant fingers can essentially help to improve the reliability and the robustness of ARMS. As the fingers are associated with pheromone values, they will be removed when their pheromones drop below a threshold value. Then the size of the routing table can be optimised.

7.2. Performance Measures

This study was designed to verify the correctness of the scheme regarding the communication continuation and to test the performance of the relocation scheme against various system parameters. Furthermore, the performance of the hybrid scheme was compared to the performance of a home-based approach and a forwarding pointer

approach. The tests had assumed a simple computational model; a sender object sent regular messages to a receiver object that changed its location according to a preset migration rate. Further, a new location was randomly chosen for unbiased tests. Because the pattern of message passing is less relevant here, the tests did not use the trace files. Rather, the message pattern was produced based on a pseudo-random generator.

As this performance study centred at the impact of object migrations on the efficiency of message passing between objects, the simulation tests focused on the following aspects.

- *Average Migration Time* – is the duration of a migration process, which is influenced by the synchronisation process and the distance of the migration. The units for the average migration time are based on a tick of the simulator and thus it neither requires a time unit nor implies the real performance of benchmark programs. Because the home-based approach needs to synchronise three parties: the migrating peer, the migrated peer and the home peer, one can expect a higher migration time in that scheme compared to other schemes.

- *Query Path Length* – refers to the expected length of a query path. With migration, the query path consists of two phases: locating and relocation. The relocation path is determined by a relocation scheme. Further, ARMS seamlessly integrates two protocols during an object search as the relocation phase is transparent to the sender object.

- *Average Invocation Time* – refers to the time between the departure of an invocation from the source and its arrival at the destination. During the object migration, an invocation may be suspended until the migration process is finished. Therefore, the migration process has a significant impact on the invocation time.

Furthermore, system variables used in the simulation tests include

- *Migration Count* – sets the total number of migrations to be processed during simulation. Thus, a simulation test is terminated when the number of migrations is satisfied;

- *Messaging Interval* – controls the rate of messages sent from a sender that has a fixed location. The smaller messaging interval results in more messages being sent by the sender at the same period; and

- *Network Size* – refers to the number of peers in a network. The network size can affect the migration time, the path length and the invocation time.

7.3. Simulation Analysis

7.3.1. Scaling with the migration counts

Here, the performance of the three relocation schemes was tested against the migration counts. The migration counts were increased from 2 to 16 at the step of two. The migration interval was set to 150 unit time while the messaging interval was 200 unit time. Four separate tests were done so as to take the average of these values. Figure 67 shows the average migration time among the three schemes. The simulation results are

coincided with the hypothesis; the home-based approach exhibited the highest migration time on average. Because the hybrid scheme did not block messages during the migration, it had the same performance as the forward pointers scheme.

Furthermore, the figure revealed that the migration time of the forward pointers and the hybrid scheme was invariant to the migration counts. Rather, the migration time was proportional to the distance between the last location and the new one. In contrast, the migration time of the home-based approach grew proportionally with the migration counts. This is because the migration rate of the mobile object was faster than the update speed of the home node as the result of the high migration overhead.

The high migration overhead is due to the significant query loads received by the home peer. The distribution of the query loads received by the home peer in each scheme is illustrated in Figure 68. The home peers became the communication bottleneck, where messages were quickly built up in the message queue. The overwhelming size of messages subsequently caused delays in performing migration. With the forward pointers or the hybrid scheme, on the contrary, the query loads were shared amongst multiple peers and hence the query loads were low at the home peer.

The average lengths of the query paths are shown in Figure 69. Firstly, it is notable that the query lengths were increased linearly with the migration count for forward pointers. This is because the forwarding chain was expanded after each migration. Secondly, the hybrid scheme takes advantage of the home peer and the updating process so that the query length can be controlled. Figure 69 reveals that the performance of the hybrid scheme was comparable to the home-based approach.

Finally, the average invocation time measures how long a remote invocation can arrive at the destination. It is ideal to minimise the invocation time to improve the efficiency of distributed computation. Figure 70 demonstrates that the hybrid scheme was superior to the forward pointers with the significantly shorter query path. The hybrid scheme performed similarly to the home-based approach at small migration counts. However, it showed a significant improvement in the invocation time at a large migration count.

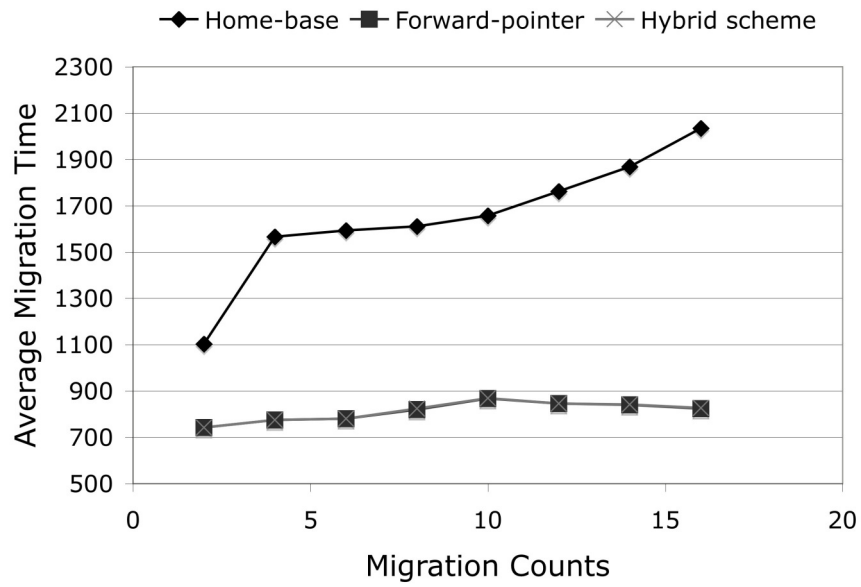

Figure 67: An illustration of the average migration time of the three schemes with respect to the migration counts.

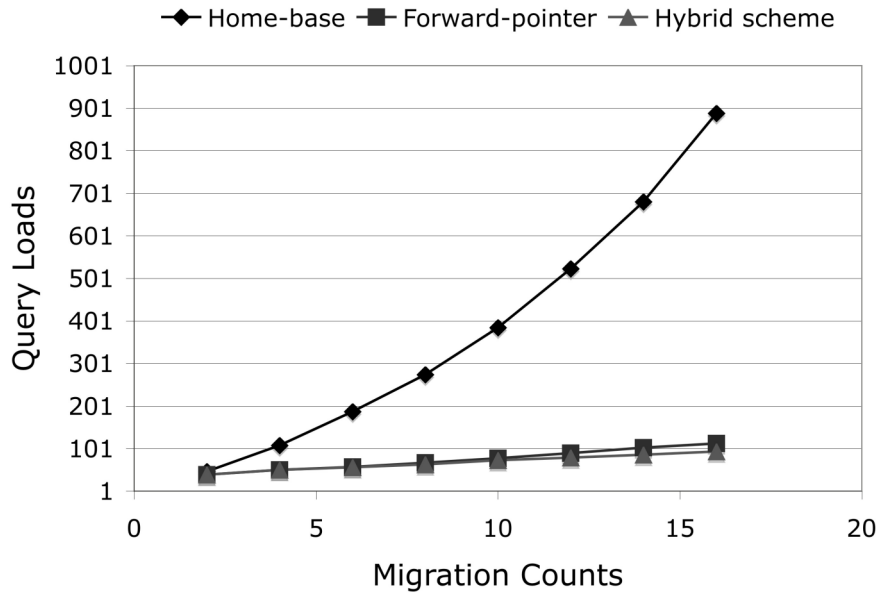

Figure 68: Distribution of the query loads received by a home peer in the function of the migration counts for each relocation scheme.

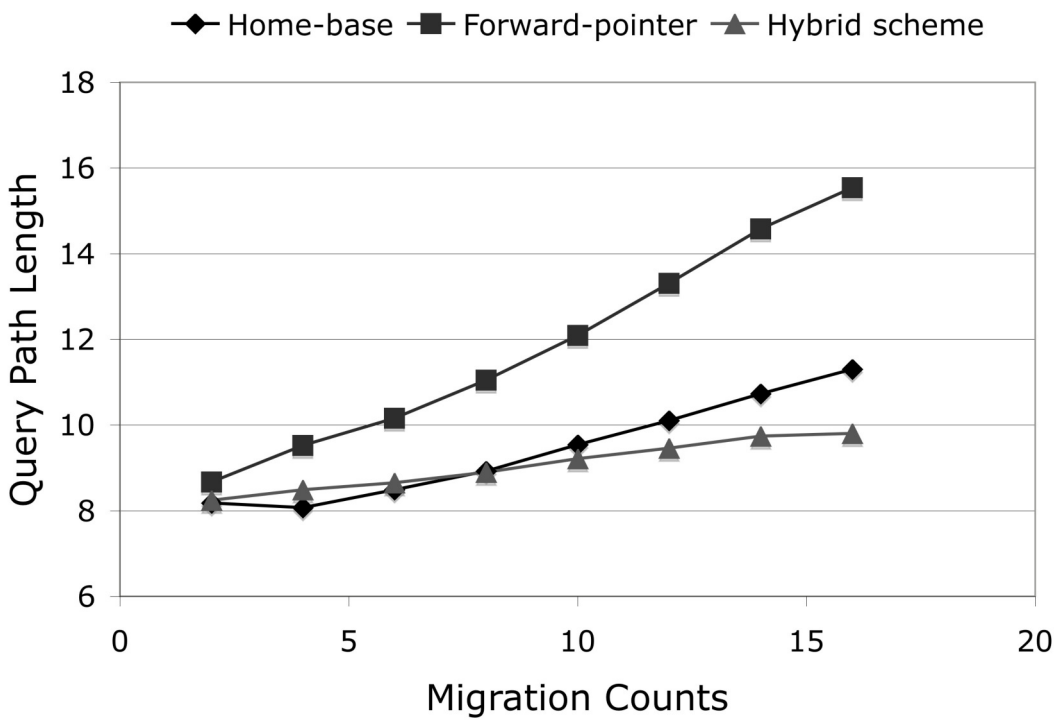

Figure 69: An illustration of the query path length of the three relocation schemes in the function of migration count.

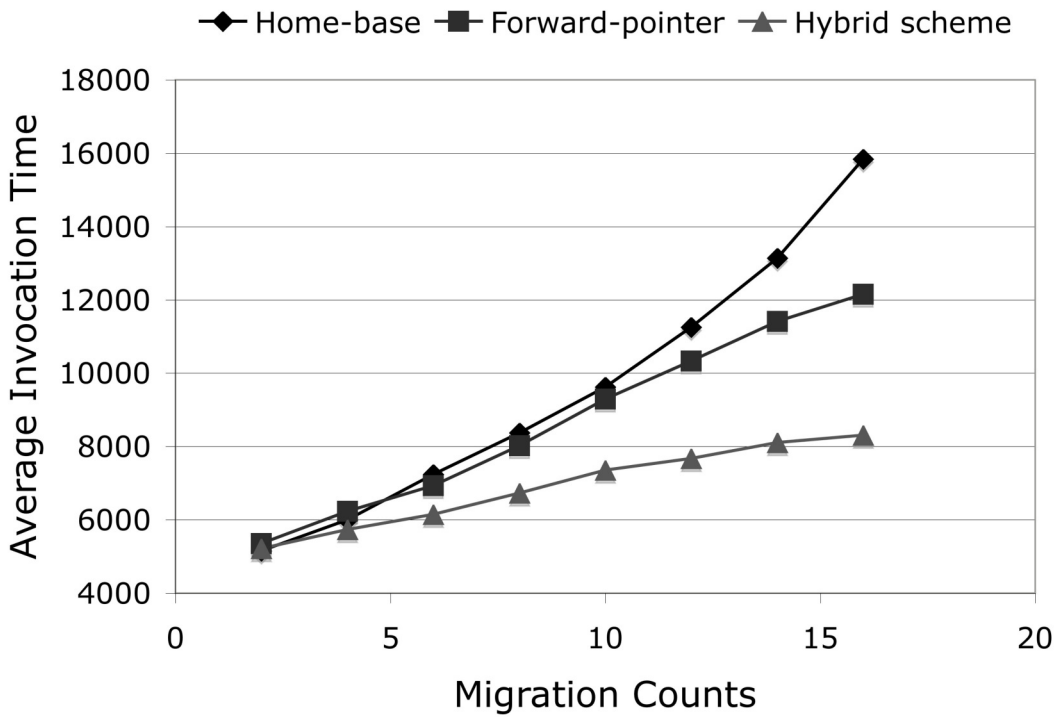

Figure 70: An illustration of the average invocation time for the three relocation schemes, as function of the migration counts.

7.3.2. Impact of messaging intervals

In the simulation tests, the messaging interval was used to control the frequency of messages sent from a sender object. The smaller interval will generate more messages at the same period. Figure 71 and Figure 72 give the performance of the average migration time and the average invocation time verses the messaging rates. In the tests, the migration counts were fixed while the messaging rate was increased from 50 unit time to 550 unit time in an increment of 50 unit time.

It clearly showed that the messaging rate did not affect the migration time of the forward pointers scheme and the hybrid scheme. However, the fast messaging frequency caused the high migration time in the home-based approach. This is again due to the performance bottleneck created by the home peer. Similarly, the invocation time decreased with the increase of the messaging rate and the three schemes showed the similar efficiency in invocation at a low messaging rate. As the profiling study showed that the majority of objects were transient objects, the short lifespan of those objects implies that they received messages within a considerably short period. Therefore, the home-based approach is not suitable to be implemented in such a system albeit it is questionable whether there is need for migrating transient objects. Nonetheless, the hybrid scheme performed outstandingly at any messaging rates.

7.3.3. Scaling with the network sizes

The third series of tests aimed to examine the scaling property of the three schemes in terms of the network size. Five sets of network sizes were tested; 100, 400, 900, 1600 and 2500 while other system variables were fixed at each simulation test.

Figure 73 showed that query path lengths for the three relocation schemes grew longer with the increase of the network size. The growth of query paths was primarily due to

the increase of the locating path, where the dashed line represents the theoretical query path as a function of the network size. The slope of the home-based approach is approximately fitted to the theoretical line. Due to the use of home peers, the hybrid scheme achieved similar results as the home-based scheme. In contrast, the incline of the forward pointers was faster than the other schemes and thus the forward pointers scheme had the poorest scaling property among the three relocation models as expected.

The growth of a network size could increase the migration time as demonstrated in Figure 74. When the network size grew, the physical distance between two peers became longer and hence the migration process took a longer time. Extra delays in the migration process caused more messages being queued at the home peer. As mentioned previously, the migration process of the home-based scheme is extremely sensitive to the query loads. Subsequently, the migration time was worsened for the home-based scheme within a large-scale system.

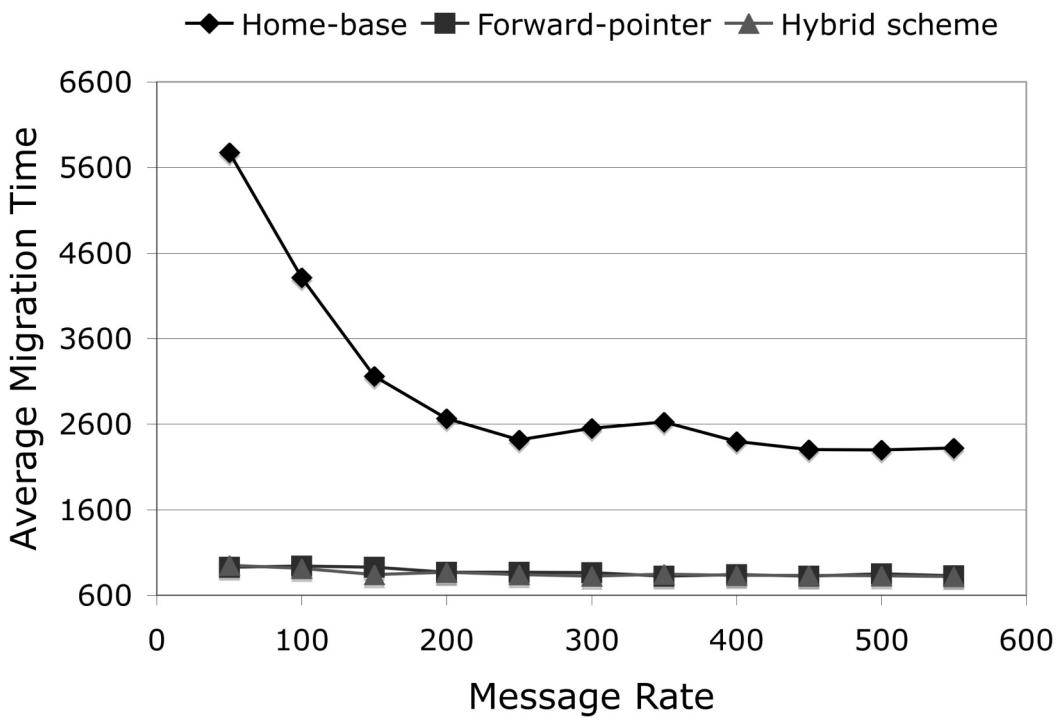

Figure 71: A demonstration of the average migration time at various messaging rates.

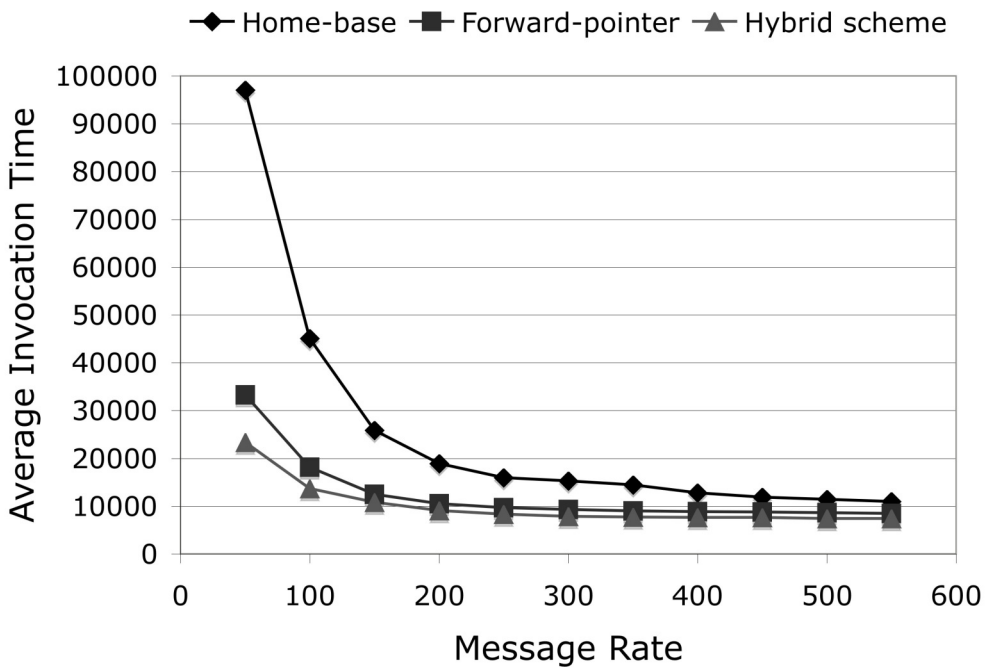

Figure 72: A demonstration of the average invocation time at various messaging rates.

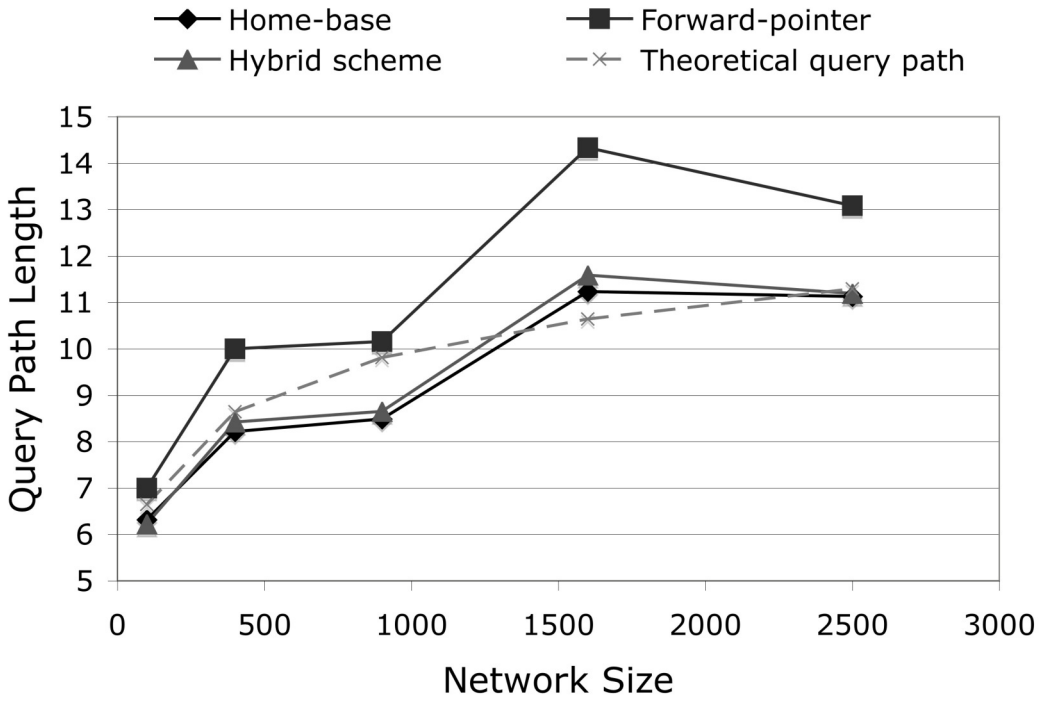

Figure 73: The query path length verses the network sizes under the different network sizes.

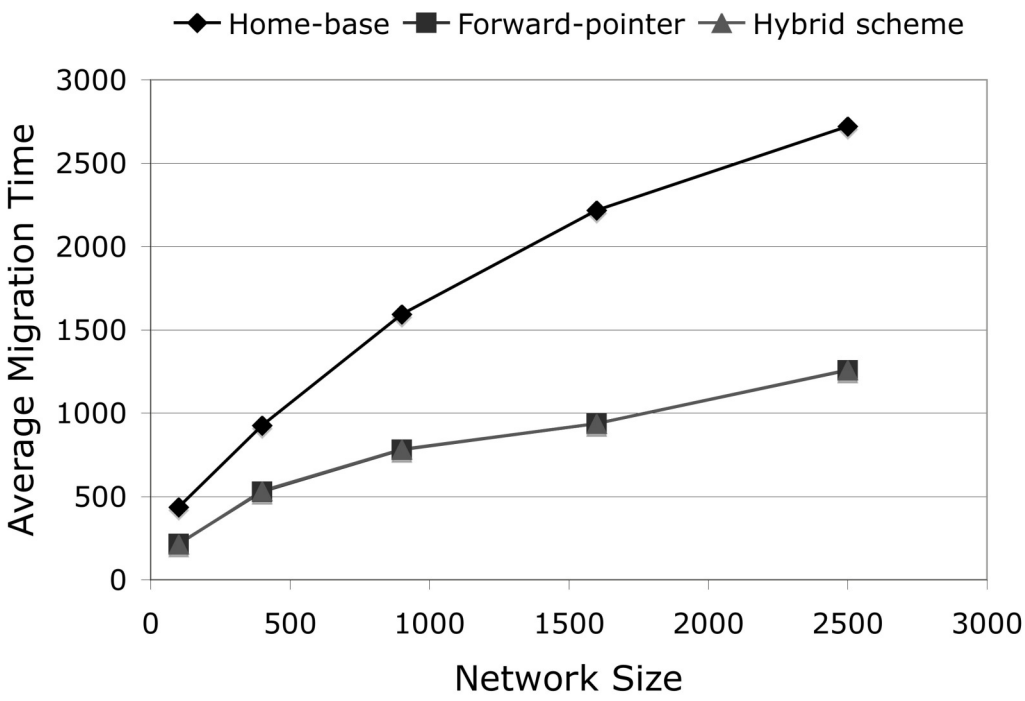

Figure 74: The migration time verses network sizes under the different network sizes.

7.4. Discussion on Simulation Results

The home approach uses the birthplace of an object as a persistent registry for keeping the current address of the migrated object. If the new address stored in the home peer is up to date, the number of steps required to locate the migrated object is $\log(N) + 1$, where N is the network size. The simulation observations were approximately fitted to the theoretical prediction. However, the home approach requires a synchronisation method to update the home peer at each migration. Hence, it incurred significant migration overheads as revealed by the simulation tests. Further, the home peer easily became a performance bottleneck and thus they were very sensitive to large workloads as the consequence of the fast message rate or the high frequency of migrations. Moreover, the trombone problem, where the migrated object and the sender are both separated by a large distance from the home peer, delayed the arrival of messages at the object's new location. Consequently, the home approach is not suitable for providing relocation for a distributed system with a high frequency of message passing or a large number of nodes.

To reduce the migration overheads in ARMS, a forward pointer is created at the old node for temporally directing messages to the new location. This approach cut down the migration time by using a simpler migration synchronisation and shedding the workloads of the home peer through the forward pointers. The forward pointers can re-route messages before they reach the home peer. The drawback of the forward pointers, however, is that the forward chain can grow linearly with the number of migrations. The long forwarding chain eventually caused high overheads in the object locating. As demonstrated in the tests, the growth of the query path was faster than the home approach with respect to the migration counts or the network sizes. As a consequence, the backtracking approach is applied in ARMS for shortening the forward chain.

When the migration is completed, an update message is sent from the new address of the object to its home peer. The update message is routed in the reverse of the forward chain so that nodes along the chain can revise their forwarding pointers to the new address. The outcome of the backtracking process is that more shortcuts to the migrated objects are deposited around the overlay. This can make messages redirect to the new location more rapidly. Thus, the average length of query paths in the hybrid scheme was the shortest as observed in tests. Further, these redundant links can be removed to optimise the size of the forwarding table when their pheromone values drop below a threshold. Finally, tests showed that the hybrid approach improved both the efficiency of migration and the length of a query path. Consequently, the average invocation time of the hybrid scheme was shorter than that of the home approach or the forward pointers.

8. Conclusions

8.1. Extensive Supports for Runtime Optimisation to Improve Routing Efficiency

The goal of this research was to design an efficient and scalable naming system that provides name services for object-based distributed computing systems. The secondary goal was to extend such a naming system to support the object migration process. These goals were achieved through a novel naming system termed ARMS (Adaptive, Randomised and Migration-enabled Scheme).

While the routing performance of ARMS – O(log*N*) – is similar to most structured naming systems such as the Chord [23] and the Tapestry [34], ARMS uses an adaptive technique to support runtime optimisation for achieving greater routing efficiency. In contrast, the rigidness of the structured naming systems restricts the routing performance when running with real world applications [40, 128]. The real applications tend to be dynamic and to exhibit uneven workloads in the message passing, leading to query hotspots and routing hotspots in the network [42]. This approach can adapt to the evolving topology of object relationships and the constantly changing network conditions. As shown in Chapters 6, the routing performance of ARMS was up to 42% shorter than that of the Chord and its name resolution process was up to 51% faster.

An ant colony optimisation (ACO) algorithm drives the adaptability of this approach. The algorithm offers a set of parameters to control the ability to explore or to exploit useful routing paths. In this study, the set of parameters was manually chosen from the results of multiple tests. In real life situations, it is not practical to select the parameters manually for every application; either, it is not feasible to use the same set of the

parameters for various applications since their workloads can be dramatically different. Consequently, an automated process to configure the parameters should be considered to use with this approach. Such a process must be able to dynamically adjust the ACO algorithm according to the runtime characteristics of the software.

Further, this study showed that not all ACO parameters affected the routing performance equally. The routing performance was sensitive to the parameters such as the evaporation rate, the pheromone importance factor and the heuristic important factor, while the impacts of the decay rate and the heuristic constant on the performance were weaker. This suggests that the complexity of the ARMS model could be simplified without too much of a sacrifice for the routing efficiency. Such a simplification will potentially benefit an automated process that configures the algorithm at runtime.

The ability to explore paths relies on the query multicasting on the randomised network. The problem, though, is that multicasting was sensitive to the increasing message workloads as a result of the growing object population. It is realistic to predict that the amount of objects and messages generated at runtime will expand in the future due to the proliferation of functionalities in software. The high volume of messages can potentially degrade the performance of this approach. A solution is to limit the number of the multicasting messages. However, using too few messages can reduce the ability to explore useful routing paths and hence decrease the adaptability of the algorithm. Alternatively, objects with intensive message workloads can be replicated or transferred to improve the communication locality of objects. The communication locality would help to reduce the overall number of messages in the system, as well as to improve the effectiveness of the ARMS routing protocol.

8.2. Seamless Integration of the Naming Process and the Routing Process

Virtually all distributed computing systems separate the naming process from the message routing process. Hence, the total delays incurred in message passing in those systems is given by $D_{total} = D_{resolution} + D_{reply} + D_{routing}$. In contrast, this approach allows a distributed system to transport messages directly to the destinations based on symbolic names. Without having to wait for the reply of the name resolution, the delay incurred in ARMS is exactly the time spent on the name resolution, i.e. $D_{total} = D_{resolution}$. This approach arguably trades increasing communication efficiency for a higher load balancing cost.

Load balancing is achieved via tightly controlled data placement in structured naming models; every node keeps roughly the same number of object identifiers and hence receives the even number of name requests. However, this approach argues the necessity of such a placement policy if the pattern of message passing is inherently unbalanced in the actual software. As observed in this study, certain objects in a program were referenced more frequently than other objects. Consequently, the owner nodes of those objects received significantly more name requests and became hotspots in the network. Even though this approach applies redundant paths to ease congestions around hotspots, ultimately it relies on a resource scheduler for managing workloads in the system.

The efficiency of name-based routing in ARMS can be affected by temporal coupling messages. Those messages can spark off a considerable amount of multicasting messages in the network as a reply embedded with the address of the destination might arrive later at the source node. This can seriously obstruct the links of the network. To

overcome the congestion problem, this study restricted the use of the multicasting for the first time of resolving an object name at a sender; every subsequent call to the same object from that sender just issued a single message based on the same name. Once the address of the object was received, the sender routed subsequent messages to the object using the address.

The discrepancy between topologies of the overlay network and the physical network can cause significant delays in the name resolution, and thereby slow down the name-based routing. This approach allows the links of the overlay network to be re-organised at runtime for reconciling with the physical network. However, the dynamic approach can incur delays in the convergence of the two networks. In contrast, other naming systems such as SkipNet [149] have emphasised on an overlay construction method that structurally optimises the proximity between nodes on the overlay. It would be useful to explore effective construction methods and to incorporate them into ARMS for assisting the dynamic restructuring approach and enhancing the performance of the name-based routing.

8.3. Provision of A Scalable Transparent Relocation Scheme for Referencing Migrated Objects

Migration is an important technique for dynamic resource management. ARMS provides a transparent relocation scheme that essentially allows migrated objects to be accessed via the same symbolic names. The key advantage of the approach over other relocation schemes [49, 138, 139] is that it ensure the time to find a migrated object varies as the logarithm of the size of the network – i.e. O(log*N*). Hence, the routing performance remains bounded regardless of whether the recipient object has been moved to a new location.

An important conclusion of the study of the relocation scheme is that frequent migration did not well suite to work for object-based distributed systems because a majority of objects were indeed transient objects. Those objects received or sent only a few messages and subsequently were discarded from the execution. In this regard, the initial position at where such an object is created appears a more valuable choice to improve the execution efficiency.

As demonstrated in the study, ARMS uses the birthplace of an object to permanently keep the latest address of the migrated object and it applies forward pointers to temporally lead messages to the new location. To update the birthplace and the forward pointers, a reply message from the new location is returned to the birthplace along the forward pointers. While this study has assumed error-free paths, in the real world situation, there is a risk of losing messages due to link or node failures. This can cause disruption in the updating process and thus broadcasting is required to restore the recent address of the migrated object stored at the birthplace. To reduce the impact of the link failures, a second message would be returned to the birthplace via a separate route. Obviously, the redundant messages might increase network workloads depending on the frequency of object migrations.

8.4. Remarks on the Java-based Benchmark Programs

In modern computing, benchmark programs play an important role in the system design and the performance evaluation across various architectures. This study has examined four Java-based programs chosen from separate software categories. It is reasonable to postulate that the chosen programs would provide sufficient information to demonstrate the key features of this approach through the simulation study. However, it could be more desirable if a set of standard benchmark programs is used in this study.

Particularly, these benchmarks should mimic workloads of message passing between objects within a sizeable population.

In addition, benchmark programs written in other object-oriented languages are crucial to analyse because the distributed object-oriented system should naturally support the efficient execution of any of the current object-oriented languages just as present von Neumann machines efficiently support the execution of existing procedural languages.

How this approach performs under mixed workloads as a result of concurrently executing multiple applications inside a system is a question worth examining. The mixed workloads can essentially demonstrate, aside from raw performance, qualities including availability, scalability and load balancing. The need for efficient support for mixed workloads might require modifying the current approach since it was simply designed for running a single application.

8.5. Improvements to the System

This study highlights the potential use of mixed heuristic rules to improve the success rate of finding the requested node, leading to a shorter forwarding path in the decentralised name resolution. There is a considerable room for improving the performance of the mixed rules since this study simply adopted an averaging approach for calculating the mixed rules. A better alternative would be a multiple-objective ACO algorithm [150, 151, 152] that can compensate conflict rules. Further, a migration policy can be incorporate into a heuristic rule so that messages can be redirected to the new locations of migrated objects faster.

The use of multicasting in query routing places a burden on the scalability of the system. It is desired to use fewer messages in the routing to reduce the network traffic,

particularly when the object communications are intensive. It is possible to utilise adaptive multicasting to dynamically control the amount of broadcasting messages based on the network conditions; it may increase the complexity of the routing algorithm. Ideally, a pathfinding scheme should discover an optimal path – not always the shortest one – by examining a small set of nodes. The scheme will also benefit the routing of temporal coupling messages issued from the same source as those messages can be converged quickly to the optimal path towards the destination.

This approach demonstrated the significance of a runtime optimisation to the efficiency of the distributed name resolution because of the dynamics of message passing and a network. However, the dynamic approach incurs significant overheads due to the iterative process. Thus, there would be an advantage in studying a preliminary approach that optimises the configurations of the system such as the parameters of the routing algorithm prior to the execution of any application. Three kinds of configurations can be benefited from such an approach. Firstly, the virtual links of the overlay network can be reorganised to correspond to the object topology. The new structure will improve the routing locality and hence reduce the latency in the decentralised name resolution. Secondly, the preliminary approach can adjust the routing parameters based on the static information of the pattern of object communication. This helps to enhance the effectiveness of the routing algorithm and to decrease the delays incurred in the dynamic process. Finally, a migration or replication policy can utilise the preliminary analysis to achieve the better performance in resource management.

This research is based on an assumption that modifying the virtual links between nodes is trivial and modifying the underlying physical links is impractical. However, the performance of the name-based routing can be substantially improved if the physical

links are dynamically programmable. This would potentially eliminate the use of a virtual topology and thus remove the latency in query forwarding due to the topology discrepancy in the two layers. Further, the programmable network can physically separate communication workloads for the uneven message passing between objects. Therefore, better load balancing can be realised in the system. If the number of links in the physical network can be dynamically expanded as the links of a virtual topology, message passing can take advantage of the smaller network radius to cut down the travelling distance between a source and a destination.

Appendix: A Discrete Event, Trace Driven Simulator for Studying Object-Based Distributed Computing Systems

Simulation played an important role in this study as it was used to investigate the characteristics of the ARMS naming model and to evaluate its performance. The simulator is essentially based on the discrete event simulation and uses trace files as inputs to reproduce the patterns of message passing between objects in a Java-based benchmark program. The design of the simulator was inspired by organisations of early message-driven systems such as J-machine [14] and Decentralised Object-Oriented Machine (DOOM) [13]. Figure 75 depicts an abstract view of a computing node defined in the simulator.

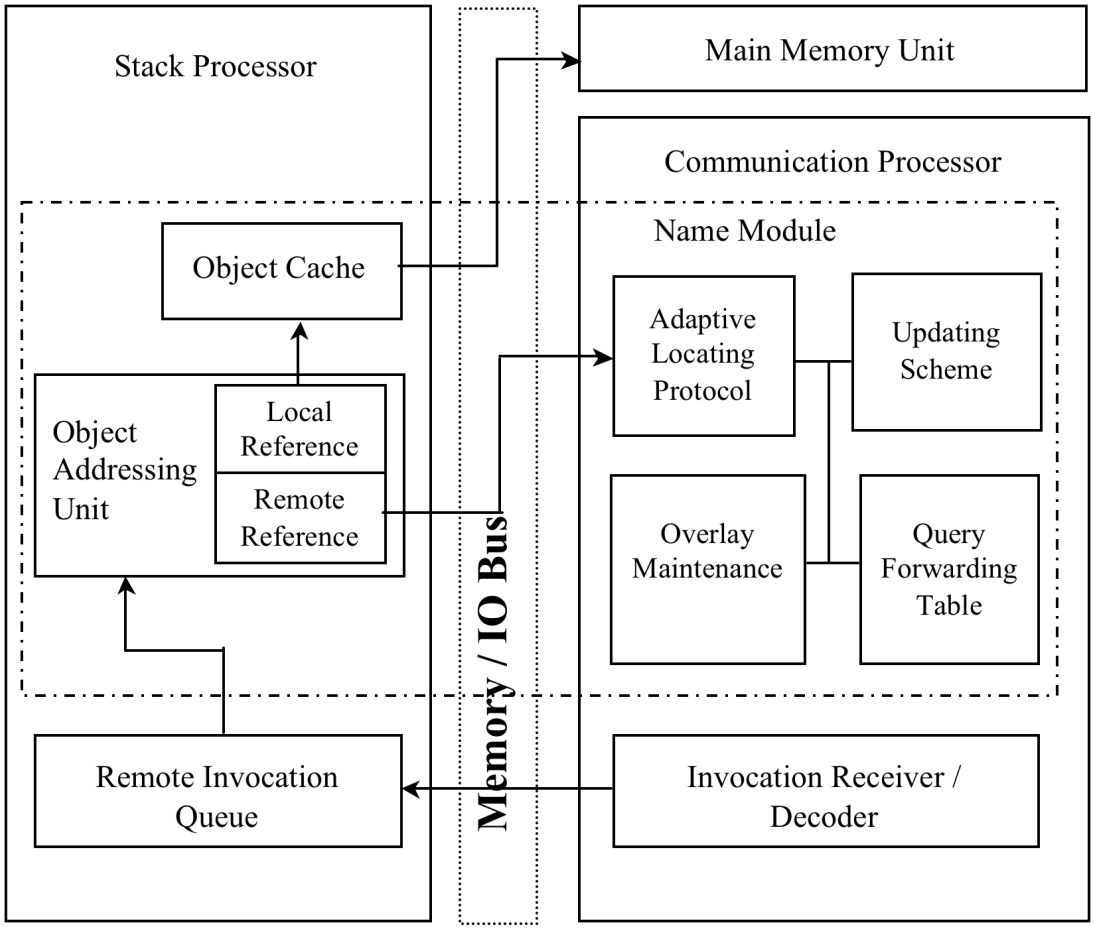

Figure 75: An abstract representation of a computing node defined in the discrete event simulator.

The notable characteristic of such systems is the tight coupling of the processing unit and the communication unit. Here, the focus of the system is the naming module contained in each computing node. Functions of the naming module are twofold: to support referencing to local objects and to provide accesses to remote objects based on transparent names. Obviously, different concerns arise from design of the two functions and thus they need to be considered separately. For instance, virtual object caches [15] have been proposed to support fast accesses to an object, including its fields and methods, created in the main memory within a machine. For the purpose of this thesis, it has concentrated on the support for referencing remote objects.

As shown in Figure 75, there are four basic components to perform the remote referencing; namely, *adaptive locating protocol*, *updating scheme*, *overlay maintenance*, and *query forwarding table*. Each component is corresponding to a functional part defined in ARMS. The overlay maintenance element controls the construction and restructuring of the ARMS overlay network. Before a new machine is inserted into the ARMS overlay, it needs to contact a list of existing machines to determine the machine's new identifier used in the Chord ring. Afterwards, the set of object identifiers assigned to the new machine's immediate neighbours needs to be resized. Depending on a load optimisation policy, object identifiers can be evenly distributed among these machines. The query forwarding table, on the other hand, contains mappings of node identifiers to their physical addresses. These mappings are used to route name queries on the name resolution.

The forwarding table is utilised by both an adaptive locating protocol and an updating mechanism as described in Chapter 3. When a remote reference is executed, the adaptive locating protocol first checks the mappings in the query forwarding table. If

there is a match of a node identifier, the message will be delivered to that node. On the other hand, if no match can be found, the protocol will nominate the next node for receiving the message. As explained earlier, the forwarding table of ARMS is dynamically managed as contradict to the static forwarding table used in structured models. The table is adjusted by the updating scheme using a reply message returned from the destination node.

Figure 76 depicts the class diagram of the key classes in the simulator. The *Node* class captures the behaviours of the computing node illustrated in Figure 75. Nodes can form different topologies: 2D mesh, 2D torus, cube, tree and the complete mesh, through *ports*. Ports are associated with *message queues* that were used for inbound and outbound communications. Moreover, the link latency is defined in a port by the random generator. For the sake of simplicity, the thesis has assumed the communication links were error-free and thus it is trivial to implement an error detection mechanism in such a communication system.

The *Object* class represents a Java object in computation. Its execution sequence is determined by a set of instrumented invocations acquired from a Java object within a benchmark program. Because local computations were filtered by the profiler, the thesis has assumed that there was a delay between every two remote invocations. Such a delay, which simply follows a normalised distribution, is meant to replicate a local computation. Furthermore, every object can be transferred to another node at runtime. This is necessary for the study of object relocating schemes in Chapter 7.

Figure 77 illustrates the workflows of activities of the *SimEngine* class, the *Node* class, and the *Object* class in the simulator.

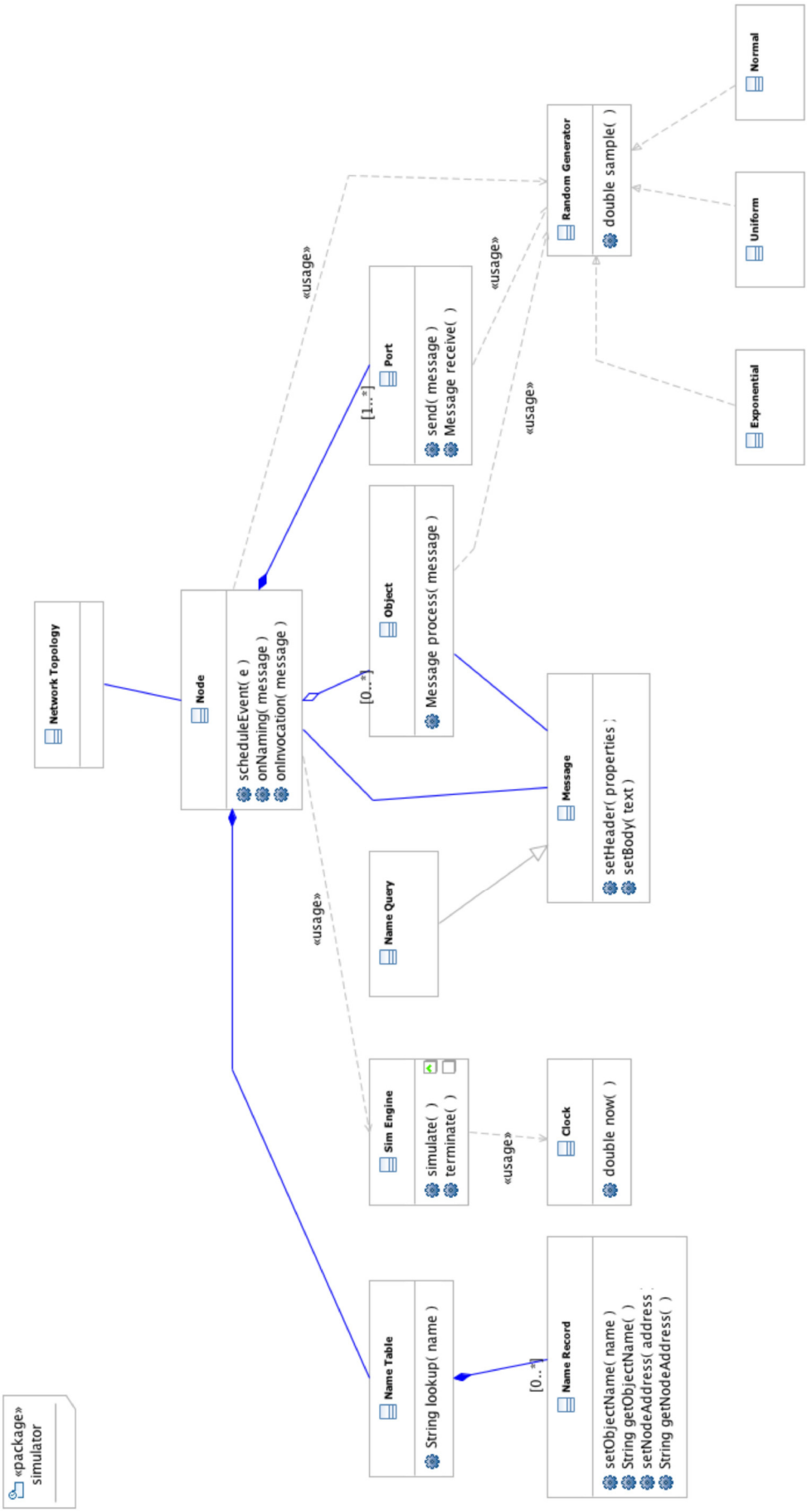

Figure 76: A UML (Unified Modelling Language) class diagram of the discrete event simulator.

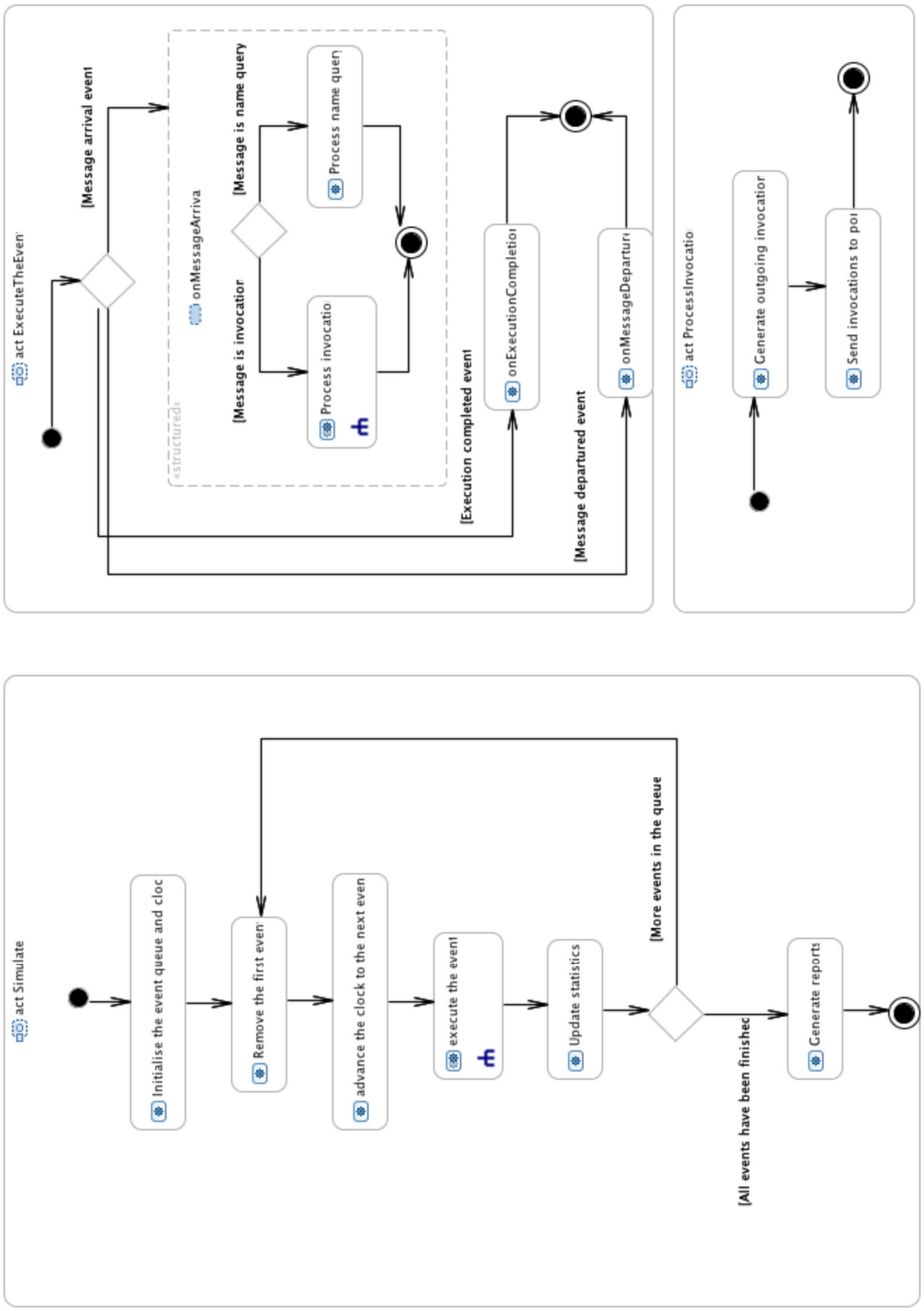

Figure 77: An activity diagram of the discrete event simulator.

Bibliography

[1] T. Korson and J. D. McGrego, "Understanding Object-Oriented: A Unifying Paradigm," *Communications of the ACM,* vol. 33, pp. 40-60, 1990.

[2] B. Eich, "JavaScript," Mozilla Foundation, 1995. Available: http://www.mozilla.org/js/.

[3] J. Gosling, B. Joy, and G. Steele, *Java Language Specification*: Sun Microsystems, Inc, 1996.

[4] Y. Matsumoto, *Programming Ruby - The Pragmatic Programmer's Guide*: Addison Wesley Longman, Inc., 2000.

[5] S. Micheloud, "Scala and AsmL Side By Side," Swiss Federal Institute of Technology Lausanne, Lausanne, May 29 2003.

[6] S. Wiltamuth and A. Hejlsberg. (2007). C# Language Specification. [Online]. Available: http://msdn2.microsoft.com/en-us/vcsharp/aa336809.aspx.

[7] "Rhino," Mozilla Foundation, 1997. Available: http://www.mozilla.org/rhino/.

[8] "Sun Java System Portal Server," Sun Microsystems Inc., 2008. Available: http://www.sun.com/software/products/portal_srvr/index.xml.

[9] "Eclipse IDE," The Eclipse Foundation, 2009. Available: http://www.eclipse.org.

[10] W. Rasband, "ImageJ," National Institute of Health, 1997. Available: http://rsb.info.nih.gov/ij/.

[11] T. Robinson, F. Tolmasky, and R. Boucher, "280 North," 280 North, Inc., 2008. Available: http://280north.com/.

[12] J. M. Pendleton, S. I. Kong, E. W. Brown, F. Dunlap, C. Marino, D. M. Ungar, D. A. Patterson, and D. A. Hodges, "A 32-bit Microprocessor for Smalltalk," *IEEE Journal of Solid-State Circuits,* vol. 21, pp. 741-749, 1986.

[13] W. J. H. J. Bronnenberg, L. Nijman, E. A. M. Odijk, and R. A. H. v. Twist, "DOOM: A Decentralized Object-Oriented Machine," *IEEE Micro,* vol. 7, pp. 52-69, 1987.

[14] W. J. Dally, "Fine-Grain Message-Passing Concurrent Computers," *The Third Conference on Hypercube Concurrent Computers and Applications: Architecture, Software, Computer Systems, and General Issues*, 1988, vol. 1, pp. 2-12.

[15] V. Narayanan, "Issues In The Design Of A Java Processor Architecture," Ph.D. thesis, Department of Computer Science and Engineering, University of South Florida, 1998.

[16] H. McGhan and J. M. O'Connor, "PicoJava: A Direct Execution Engine For Java Bytecode " *IEEE Computer,* vol. 31, pp. 22-30, 1998.

[17] A. Krall, A. Ertl, and M. Gschwind, "JavaVM Implementation: Compilers Versus Hardware," in *Computer Architecture*: Springer, 1998, pp. 101-110.

[18] M. Tremblay, J. Chan, S. Chaudhry, A. W. Conigliam, and S. S. Tse, "The MAJC Architecture: A Synthesis Of Parallelism And Scalability," *IEEE Micro,* vol. 20, pp. 12-25, 2000.

[19] M. Schoeberl, "JOP: A Java Optimized Processor for Embedded Real-Time Systems," PhD thesis, Vienna University of Technology, 2005.

[20] G. Wright, M. L. Seidl, and M. Wolczko, "An Object-Aware Memory Architecture," Sun Microsystems Inc., SMLI TR-2005-143, 2005.

[21] M. Schoeberl, "Architecture For Object Oriented Programming Languages," *Proceedings of The 5th International Workshop on Java Technologies for Real-Time and Embedded Systems*, Vienna, Austria, 2007.

[22] M. Schoeberl, "A Java Processor Architecture for Embedded Real-Time Systems," *Journal of Systems Architecture: the EUROMICRO Journal,* vol. 54, pp. 265-286, 2008.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *The 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, United States, 2001, pp. 149-160.

[24] A. Goldberg and D. Robson, *Smalltalk-80: The Language and Its Implementation*: Addison-Wesley, 1983.

[25] (2004). Common Object Request Broker Architecture: Core Specification. [Online]. Available: http://www.omg.org/spec/CORBA/3.1/.

[26] (2009). An Overview of RMI Application. *The Java Tutorials* [Online]. Available: http://java.sun.com/docs/books/tutorial/rmi/overview.html.

[27] (2009). .NET Remoting Overview. *Microsoft Developer Network* [Online]. Available: http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx.

[28] A. G. Fraser, "On the Meaning of Names in Programming Systems," *Communications of the ACM,* vol. 14, pp. 409-416, 1971.

[29] R. S. Fabry, "Capability-based Addressing," *Communications of the ACM,* vol. 17, pp. 401-412, 1974.

[30] J. Shoch, "Inter-network Naming, Addressing, and Routing," *the Seventeenth IEEE Conference on Computer Communication Networks*, 1978, pp. 72-79.

[31] B. M. Hauzeur, "A Model for Naming, Addressing, and Routing," *ACM Transactions on Information Systems,* vol. 4, pp. 293-311, 1986.

[32] J. Postel, "Internet Name Server," 1979. Available: http://www.isi.edu/in-notes/ien/ien116.txt.

[33] P. Mockapetris and K. J. Dunlap, "Development of the Domain Name System," *ACM SIGCOMM Computer Communication Review,* vol. 18, pp. 123-133, 1988.

[34] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," University of California, Berkeley, UCB/CSD-01-1141, 2001.

[35] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing In A Small World," *the Fourth USENIX Symposium on Internet Technologies and Systems*, Seattle, United States, 2003.

[36] "Kazaa," Sharman Networks, 2001. Available: http://www.kazaa.com/.

[37] (2008). Gnutella's Official Website. [Online]. Available: http://www.gnutella.com/.

[38] S. Rowstron and P.Druschel, "Pastry: Scalable, Distributed Object Location And Routing for Large-Scale Peer-to-Peer Systems," *IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, 2001, pp. 329-350.

[39] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *ACM SIGCOMM*, San Diego, United States, 2001, pp. 161-172.

[40] P. Keleher, B. Bhattacharjee, and B. Silaghi, "Are Virtualized Overlay Networks Too Much of a Good Thing," *Peer-to-Peer Systems: First International Workshop, IPTPS,* vol. 2429: Springer, 2002, pp. 225-231.

[41] M. Castro, P. Druschel, Y. C. Hu, and A. I. T. Rowstron, "Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks," Microsoft Research, MSR-TR-2002-82, 2002.

[42] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing Algorithms for DHTs: Some Open Questions," *International Workshop on Peer-to-Peer Systems (IPTPS),* vol. 2429, 2002, pp. 45-52.

[43] K. Gummadi, R. Gummadi, S. Gribble, and S. Ratnasamy, "The Impact of DHT Routing Geometry on Resilience and Proximity," *the 2003 Conference on Applications, Technologies, Architectures, And Protocols for Computer Communications*, Karlsruhe, Germany, 2003, pp. 381-394.

[44] G. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems," PhD thesis, Massachusetts Institute of Technology, 1985.

[45] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Computing Surveys,* vol. 32, pp. 241-299, 2001.

[46] J. H. Saltzer, "Naming and Binding of Objects," in *An Advance Course on Operating Systems,* vol. 60, M. J. Flynn, J. Gray, A. K. Jones, K. Lagally, H. Opderbeck, G. J. Popek, B. Randell, J. H. Saltzer, and H. Wiehle, Eds. London: Springer-Verlag, 1978, pp. 99-208.

[47] B. Bayerdorffer, "An Analytical Taxonomy of Naming Systems," University of Texas, Austin, TR-92-48, 1992.

[48] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A Layered Naming Architecture for the Internet," *the 2004 Conference on Application, Technologies, Architectures, and Protocols for Computer Communications*, Portland, USA, 2004, pp. 343-352.

[49] K. E. Falkner, "The Provision of Relocation Transparency Through a Formalised Naming System in A Distributed Mobile Object System," PhD thesis, Department of Computer Science, University of Adelaide, 1999.

[50] (2000). Trading Object Service Specification. [Online]. Available: http://www.omg.org/docs/formal/00-06-27.pdf.

[51] D. E. Corner and L. L. Peterson, "Names and Name Resolution," in *Concurrency Control and Reliability in Distributed Systems*: Van Nostrand Reinhold Co., 1987, pp. 489-524.

[52] J. Saltzer. (1993). On The Naming and Binding for Network Destinations. [Online]. Available: http://www.rfc-archive.org/getrfc.php?rfc=1498.

[53] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor Information Networking Architecture and Applications," *IEEE Personal Communications,* vol. 8, pp. 52-59, 2001.

[54] J. C. Browne, M. Yalamanchi, K. Kane, and K. Sankaralingam, "General Parallel Computations on Desktop Grid and P2P Systems," *Proceedings of The Seventh Workshop on Languages, Compilers, and Run-Time Support for Scalable Systems*, 2004, vol. 81, pp. 1-8.

[55] M. Bowman, S. K. Debray, and L. L. Peterson, "Reasoning About Naming Systems," *ACM Transactions on Programming Languages and Systems (TOPLAS),* vol. 15, pp. 795-825, 1993.

[56] H. Zimmermann, "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications,* vol. 28, pp. 425-432, 1980.

[57] C. M. Bowman, "Univers: The Construction of An Internet-Wide Descriptive Naming System," PhD thesis, The University of Arizona, 1990.

[58] L. L. Peterson, "The Profile Naming Service," *ACM Transactions on Computer Systems,* vol. 6, pp. 341-364, 1988.

[59] D. B. Terry, "Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments," PhD thesis, University of California, Berkeley, 1985.

[60] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Theory of Computing Systems,* vol. 32, pp. 241-280, 1999.

[61] M. d. Rey. (1981). DARPA Internet Program Protocol Specification. [Online]. Available: http://www.ietf.org/rfc/rfc0791.txt.

[62] D. B. Terry, M. Painter, D. W. Riggle, and S. Zhou. (1984). The Berkeley Internet Name Domain Server. [Online]. Available: http://techreports.lib.berkeley.edu/accessPages/CSD-84-182.html.

[63] P. Mockapetris. (1987). Domain Names – Concepts and Facilities. [Online]. Available: http://www.faqs.org/rfcs/rfc1034.html.

[64] P. Mockapetris. (1987). Domain Names – Implementation and Specification. [Online]. Available: http://www.faqs.org/rfcs/rfc1035.html.

[65] J. Postel. (1994). Domain Name System Structure and Delegation. [Online]. Available: http://www.faqs.org/rfcs/rfc1591.html.

[66] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," *Revised Papers From The First International Workshop On Peer-to-Peer Systems,* vol. 2429: Springer Berlin, 2002, pp. 155-165.

[67] M. v. Steen, P. Homburg, and A. S. Tanenbaum, "Globe: A Wide-Area Distributed System," *IEEE Concurrency,* vol. 7, pp. 70-78, 1999.

[68] (1995). DCE Technology. [Online]. Available: http://www.opengroup.org/dce/.

[69] (1988). RPC: Remote Procedure Call Protocol Specification Version 2. [Online]. Available: http://tools.ietf.org/html/rfc1057.

[70] (2004). Naming Service Specification. [Online]. Available: http://www.omg.org/technology/documents/formal/naming_service.htm.

[71] (1999). Java Naming and Directory Interface – Service Provider Interface. [Online]. Available: http://java.sun.com/j2se/1.5.0/docs/guide/jndi/spec/spi/spicover.frame.html.

[72] (1999). Java Naming and Directory Interface – Application Programming Interface. [Online]. Available: http://java.sun.com/j2se/1.5.0/docs/guide/jndi/spec/jndi/index.html.

[73] (2010). Active Directory Architecture. [Online]. Available: http://technet.microsoft.com/en-us/library/bb727030.aspx.

[74] M. Wahl, T. Howes, and S. Kille. (1997). Lightweight Directory Access Protocol. [Online]. Available: http://www.faqs.org/rfcs/rfc2251.html.

[75] C. Weider, J. Reynolds, and S. Heker. (1992). Technical Overview of Directory Services Using the X.500 Protocol. [Online]. Available: http://www.ietf.org/rfc/rfc1309.txt.

[76] I. Foster and C. Kesselman, "Globus: a Metacomputing Infrastructure Toolkit," *International Journal of High Performance Computing Applications,* vol. 11, pp. 115-128, 1997.

[77] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," Springer Berlin / Heidelberg, 1998.

[78] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. R. Jr., "Legion: The Next Logical Step Toward a Nationwide Virtual Computer," CS-94-21, 1994. Available: http://www.cs.virginia.edu/~legion/papers/CS-94-21.pdf.

[79] M. Lewis and A. Grimshaw, "The Core Legion Object Model," *Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996, p. 551.

[80] A. S. Grimshaw and W. A. Wulf, "The Legion Vision of A Worldwide Virtual Computer," *Communications of the ACM,* vol. 40, pp. 39-45, 1997.

[81] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing*, Portland, OR, USA, 1997, pp. 365-375.

[82] I. Foster and N. T. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems," *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 1998, pp. 1-11.

[83] N. T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," 2002. Available: http://arxiv.org/abs/cs/0206040v1.

[84] G. Allen, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus," *ACM/IEEE Supercomputer 2001*, 2001, p. 52.

[85] "PVM – Parallel Virtual Machine," 2008. Available: http://www.csm.ornl.gov/pvm/.

[86] V. Ramasubramanian and E. G. Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," *Applications, Technologies, Architectures, and Protocols for Computer Communication*, Porland, Oregon, USA, 2004, pp. 331-342.

[87] (2000). The Gnutella Protocol Specification v0.4. [Online]. Available: www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[88] I. Clarke, "Freenet," The Freenet Project, 2000. Available: http://freenetproject.org/.

[89] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," *the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[90] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," 2003. Available: http://www.cs.wpi.edu/~goos/Teach/cs4513-d05/papers/p2p-tutorial.pdf.

[91] D. Doval and D. O'Mahony, "Overlay Networks: A Scalable Alternative for P2P," *IEEE Internet Computing,* vol. 7, pp. 79-82, 2003.

[92] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Survey and Tutorial,* 2004.

[93] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Designing Privacy Enhancing Technologies*: Springer-Verlag, 2001, pp. 46-66.

[94] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination," *the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, 2001.

[95] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," International Conference on Distributed Computing Systems, 2002.

[96] Q. Lv, P. Cao, and E. Cohen, "Search and Replication in Unstructured Peer-to-Peer Networks," in *ACM ICS '02*, 2002.

[97] (2004). The FastTrack Protocol. [Online]. Available: http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?view=markup&content-type=text%2Fvnd.viewcvs-markup&revision=HEAD.

[98] "iMesh," IMesh, Inc., 1999. Available: http://www.iMesh.com/.

[99] P. Ganesan, Q. Sun, and H. Garcia-Molina, "YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology," *the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, 2003, vol. 2, pp. 1250-1126-.

[100] X. Y. Zhang, Q. Zhang, Z. Zhang, G. Song, and W. Zhu, "A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance," *IEEE Journal on Selected Areas in Communications,* vol. 22, 2004.

[101] O. Sandberg, "Distributed Routing in Small-World Networks," *Algorithm Engineering and Experiments*, SIAM, 2005.

[102] A. Kumar, J. Xu, and E. W. Zegura, "Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks," *the Twenty-Four Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, 2005.

[103] N. Sarshar, P. O. Boykin, and V. P. Roychowdhury, "Scalable Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable," *the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, 2006.

[104] H. Zhang, A. Goel, and R. Govindan, "Incrementally Improving Lookup Latency in Distributed Hash Table Systems," in *ACM SIGMETRICS 2003*, 2003.

[105] S. Apel and E. Buchmann, "Biology-Inspired Optimisations of Peer-to-Peer Overlay Networks," *Praxis der Informationsverarbeitung und Kommunikation,* vol. 28, pp. 199-205, 2005.

[106] C. C. Wang and K. Harfoush, "RASTER: A Light-Weight Routing Protocol to Discover Shortest Overlay Routes in Randomized DHT Systems," *12th International Conference on Parallel and Distributed Systems*, 2006.

[107] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," *the Thirty-Second Annual ACM Symposium on Theory of Computing*, Ortland, Oregon, United States, 2000, pp. 163-170.

[108] (2008). Gnutella Protocol Specification. [Online]. Available: http://wiki.limewire.org/index.php?title=GDF.

[109] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," in *ACM CIKM '02*, 2002.

[110] Z. Zhuang, y. Liu, L. Xiao, and L. M. Ni, "Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks," *International Conference on Parallel Processing*, Kaohsiung, Taiwan, 2003.

[111] V. V. Dimakopoulos and E. Pitoura, "Performance analysis of distributed search in open agent systems," *International of Parallel and Distributed Processing Symposium*, 2003.

[112] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," *the Third International Conference on Peer-to-Peer Computing*, 2003, pp. 102-109.

[113] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," *Physical Review E.,* vol. 64, 2001.

[114] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like P2P Systems Scalable," *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2003.

[115] O. Babaoglu, H. Meling, and a. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems," *the Twenty-Second International Conference on Distributed Computing Systems*, 2002, pp. 15-22.

[116] B. F. Cooper and H. Garcia-Molina, "Ad Hoc, Self-Supervising Peer-to-Peer Search Networks," *ACM Transactions on Information Systems,* vol. 23, pp. 169-200, 2005.

[117] H. Chen, A. Gong, and Z. Huang, "Self-Learning Routing in Unstructured P2P Network," *International Journal of Information Technology,* vol. 11, pp. 59-67, 2005.

[118] E. Michlmayr, "Ant Algorithms for Search in Unstructured Peer-to-Peer Networks," *Twenty-Second International Conference on Data Engineering Workshops*, 2006, p. 142.

[119] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "A Framework for Peer-to-Peer Lookup Services Based on *k*-ary Search," Department of Microelectronics and Information Technology, Royal Institute of Technology, Kista, Sweden 2002.

[120] Y. Zhao, "Decentralized Object Location and Routing: A New Networking Paradigm," PhD thesis, University of California, Berkeley, 2004.

[121] Y. Rekhter and T. Li. (1993). An Architecture for IP Address Allocation with CIDR. [Online]. Available: http://www.ietf.org/rfc/rfc1518.txt.

[122] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. Kubiatowicz, "Approximate Object Location and Spam Filtering on Peer-to-Peer Systems," *ACM/IFIP/USENIX International Middleware Conference*, 2003.

[123] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," *ACM SIGOPS Operating Systems Review,* vol. 35, pp. 188-201, 2001.

[124] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The Design of a Large-Scale Event Notification Infrastructure," *Networked Group Communication,* vol. 2233/2001: Springer Berlin / Heidelberg, 2001, pp. 30-34.

[125] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, pp. 654-663.

[126] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A Scalable And Dynamic Emulation of The Butterfly," *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, 2002, pp. 183-192.

[127] L. Barrière, P. Fraigniaud, L. Narayanan, and J. Opatrny, "Dynamic Construction of Bluetooth Scatter Nets of Fixed Degree and Low Diameter," i*Proceedings of The Fourteenth Annual ACM-SIAM Symposium on Discrete algorithms*, 2003, pp. 781-790.

[128] X. Li and C. G. Plaxton, "On Name Resolution in Peer-to-Peer Networks," *the Second ACM International Workshop on Principle of Mobile Computing*, Toulouse, France, 2002, pp. 82-89.

[129] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Peer-to-Peer Systems*: Springer-Verlag, 2002, pp. 53-65.

[130] S. Milgram, "The Small-World Problem," *Psychology Today,* vol. 1, pp. 60-67, 1967.

[131] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for Low Latency and High Throughput," *First Conference on Symposium on Networked Systems Design and Implementation*, San Francisco, California, USA, 2004, pp. 7-7.

[132] G. S. Manku, M. Naor, and U. Wieder, "Know thy Neighbour's Neighbour: the Power of Lookahead in Randomised P2P Networks," *the 36th ACM Symposium on Theory of Computing*, 2004.

[133] N. J. A. Harvey, J. Dunagan, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties," 2002.

[134] W. Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees," *Communications of the ACM,* 1990.

[135] M. Dorigo, M. Birattari, and T. Stützle, "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique," *IEEE Computational Intelligence Magazine,* vol. 1, pp. 28-39, 2006.

[136] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive Feedback As A Search Strategy," Politecnico di Milano 1991.

[137] T. Stützle and H. Hoos, "MAX-MIN Ant System and Local Search For The Travelling Salesman Problem," *IEEE International Conference on Evolutionary Computation*, 1997, pp. 309-314.

[138] P. T. Wojciechowski, "Algorithms for Location-Independent Communication between Mobile Agents," *AISB '01 Symposium on Software Mobility and Adaptive Behaviour*, 2001, pp. 10-19.

[139] A. Di Stefano and C. Santoro, "Locating Mobile Agents In A Wide Distributed Environment," *IEEE Transactions on Parallel and Distributed Systems,* vol. 13, pp. 844-864, 2002.

[140] M. J. Demmer and M. P. Herlihy, "The Arrow Distributed Directory Protocol," *the Twelfth International Symposium of Distributed Computing (DISC'98)*, 1998, vol. 1499, pp. 119-133.

[141] E. Jul, H. Levy, N. Hutchinson, and A. Black, "Fine-Grained Mobility in the Emerald System," *ACM Transactions on Computer Systems,* vol. 6, pp. 109-133, 1988.

[142] "AutoFocus," Aduna Software Company, 2006. Available: https://wiki.aduna-software.org/confluence/display/AF/Aduna+AutoFocus+Development.

[143] "DynamicJava," Da Vinci Machine Project, 2006. Available: http://wikis.sun.com/display/mlvm/DynamicJava.

[144] (2008). Java Plug-in Guide. [Online]. Available: http://java.sun.com/javase/6/docs/technotes/guides/plugin/developer_guide/contents.html.

[145] A. Wilcox, "Java Interactive Profiler," SourceForge.net, 2006. Available: http://sourceforge.net/projects/jiprof.

[146] S. Vauclair, "Extensible Java Profiler," SourceForge.net, 2005. Available: http://ejp.sourceforge.net/.

[147] (2003). Java 2 SDK, Standard Edition Documentation. [Online]. Available: http://java.sun.com/j2se/1.4.2/docs/index.html.

[148] (2003). Tuning Garbage Collection with the 5.0 java Virtual Machine. [Online]. Available: http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html.

[149] N. J. A. Harvey, J. Dunagan, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties," *USENIX Symposium on Internet Technologies and Systems* 2003.

[150] K. Doerner, W. Gutjahr, R. Hartl, C. Strauss, and C. Stummer, "Pareto Ant Colony Optimization with ILP Preprocessing in Multiobjective project portfolio selection," *European Journal of Operational Research,* vol. 171, pp. 830-841, 2006.

[151] C. Garcia-Martinez, O. Cordon, and F. Herrera, "A Taxonomy And an Empirical Analysis of Multiple Objective Ant Colony Optimization Algorithms for The bi-criteria TSP," *European Journal of Operational Research,* vol. 180, pp. 116-148, 2007.

[152] S.-T. Lo, R.-M. Chen, Y.-M. Huang, and C.-L. Wu, "Multiprocessor System Scheduling With Precedence And Resource Constraints Using An Enhanced Ant Colony System," *Expert Systems with Applications,* vol. 34, pp. 2071-2081, 2008.

[153] A. Celesti, M. Villari, and A. Puliafito, "Design of a cloud naming framework," In *Proceedings of the 7th ACM international conference on Computing frontiers* (CF '10), New York, NY, USA, 105-106, 2010.

[154] C. Gong, J. Liu, Q. Zhang, H. Chen, Z. Gong, "The Characteristics of Cloud Computing, " In *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW)*, San Diego, CA, USA, 275-279, 2010.

[155] L. J. Zhang, Q. Zhou, "CCOA: Cloud Computing Open Architecture," In *Proceedings of IEEE International Conference on Web Services (ICWS)*, Los Angeles, CA, USA, 607-616, 2009.

[156] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, C. Dorai, "Are Clouds Ready for Large Distributed Applications," *ACM SIGOPS Operating Systems Review*, Volume 44, Issue 2, April 2010.